STRING HANDLING IN ATARI BASIC

The major difference between Microsoft and Atari BASIC is in the handling of string variables. Here is an overview of the Atari approach to strings, and a comparison with the Microsoft method.

It is often necessary to split strings into pieces called substrings. In Microsoft BASIC, this is accomplished with special functions, MID$, RIGHT$ and LEFT$. In Atari BASIC, strings are split easily by using a subscript on the string variable. For example, A$(5,10) results in a substring which starts at the fifth character of A$ and ends at the tenth character. If only one number is given in the subscript, the substring will start with that character and end with the last character of the string.

Here is a table of the Atari equivalents of Microsoft string functions:

Microsoft	Atari
MID$(A$,X,Y)	A$(X,Y)
LEFT$(A$,X)	A$(1,X)
RIGHT$(A$,X)	A$(LEN(A$)-X)

The function LEN(A$) is the same in both types of BASIC, and returns the length, or number of characters (including blanks) of the string A$. This function is also used in concatination of strings, that is, putting two strings together into one string. In Microsoft, concatenation is accomplished with a plus sign. In Atari, the second string is concatenated to the first by making it a substring which starts just after the last character. Here is an example of two types of concatenation in both BASICs:

Microsoft	Atari
A$=A$+B$	A$(LEN(A$)+1)=B$
C$=A$+B$	C$=A$ C$(LEN(C$)+1)=B$

In Microsoft, the subscript on the string indicates a string array, which is handled just like a numeric array. In Atari BASIC, however, a string array is kept in a very long string, which is put together using concatenation, and taken apart with string splitting, as shown above. Here is an example of a simple string array in both types of BASIC:

MICROSOFT	ATARI
A$(1)="AAA"	ARRAY$="AAABBBCCC"
A$(2)="BBB"	ARRAY$(1,3)="AAA"
A$(3)="CCC"	ARRAY$(4,6)="BBB"
	ARRAY$(7,9)="CCC"

It is often helpful to make all of the substrings in the array the same length, so that it is easy to calculate the position in the array. This can be done by padding the smaller substrings with blanks. Remember, blank spaces count in the length of the string.

EXAMPLE : INVERTING A STRING

```
10 DIM X$(50)
20 INPUT X
30 X$=STR$(X)
40 FOR I=1 TO LEN(X$)
50 X$(I,I)=CHR$(ASC(X$(I,I))+128)
60 NEXT I:REM THIS LOOP INVERTS
65 REM EACH CHARACTER OF STRING
70 PRINT X,X$
80 GOTO 20
```

```
10 REM 'STRARRAY' WBB 3-18-81
11 REM
12 REM THIS PROGRAM DEMOS THE USE OF
13 REM ONE STRING VARIABLE AS A
14 REM STRING ARRAY VIA SUBSCRIPTING.
15 REM
20 REM THE PROGRAM WILL HANDLE 100
21 REM 30-BYTE CUSTOMER NAMES.
22 REM
100 REM RESERVE MEMORY FOR VARIABLES.
105 REM ****************************
110 DIM ARRAY$(100*30),NAME$(30),BLANK$(30),YN$(1)
115 REM
117 REM DESCRIPTION OF STRING VARIABLES
120 REM ********************************
121 REM ARRAY$ HOLDS 100 NAMES
122 REM NAME$  ACCEPTS INPUT FOR NAME
124 REM BLANK$ USED TO BLANK FILL
125 REM YN$    ACCEPTS YES-NO ANSWERS
126 REM
140 REM CURRENT VARIABLE LENGTHS
141 REM **************************
142 REM LEN(ARRAY$)=0
143 REM LEN(NAME$)=0
144 REM LEN(BLANK$)=0
145 REM
150 BLANK$="                              "
151 REM LEN(BLANK$)=30
152 REM
200 REM INITIALIZE THE ARRAY
201 REM ********************
202 REM BY INITIALIZING THE ARRAY, MEMORY IS AUTOMATICALLY
203 REM ALLOCATED FOR THE ARRAY. THIS IS NECESSARY IN ORDER TO
204 REM ACCESS ANY SECTION OF THE STRING ARRAY. IF YOU TRY TO
206 REM ACCESS BEYOND THE CURRENT LENGTH OF THE STRING AN ERROR 5 WILL RESULT.
209 REM
210 FOR I=1 TO 100
220 ARRAY$(I*30-29,I*30)=BLANK$
230 NEXT I
240 REM LEN(ARRAY$)=3000
300 REM
301 REM BEGIN INPUT LOOP
302 REM ****************
303 REM INPUT CUSTOMER # AND VERIFY IT DOESNT ALREADY EXIST
304 REM
305 PRINT
310 PRINT "CUSTOMER NUMBER (1-100)(0=END) ";
320 INPUT I
325 IF I=0 THEN 500
330 IF ARRAY$(I*30-29,I*30)=BLANK$ THEN 400
340 PRINT "CUSTOMER NUMBER ";I;" IS ASSIGNED TO:"
350 PRINT ARRAY$(I*30-29,I*30)
355 PRINT
360 PRINT "DO YOU WISH TO REPLACE"
365 PRINT "WITH A NEW NAME (Y/N) ";
370 INPUT YN$
380 IF YN$="N" THEN 300
390 IF YN$<>"Y" THEN 360
395 REM
400 REM INPUT CUSTOMER NAME
405 REM ******************
410 PRINT "CUSTOMER NAME ";
420 INPUT NAME$
```

```
430 REM ******************************
431 REM LEN(NAME$) IS THE NUMBER OF
432 REM CHARACTERS INPUT BY THE USER
433 REM UP TO 30.  IF < 30 CHARACTERS
434 REM WERE INPUT BY THE USER, IT IS
435 REM NECESSARY TO BLANK FILL NAME$
436 REM TO 30.
438 REM ******************************
439 REM
440 NAME$(LEN(NAME$)+1)=BLANK$
450 REM
451 REM PUT NAME IN ARRAY
452 REM ****************
460 ARRAY$(I*30-29,I*30)=NAME$
470 GOTO 300
480 REM
500 REM ROUTINE TO PRINT CUSTOMER LIST
505 REM ******************************
510 PRINT
520 PRINT "DO YOU WANT TO PRINT THE CUSTOMER"
525 PRINT "LIST TO THE SCREEN (Y/N) ";
530 INPUT YN$
531 IF YN$="N" THEN 540
532 IF YN$<>"Y" THEN 510
533 PRINT "NUMBER","CUSTOMER NAME"
534 PRINT
535 FOR I=1 TO 100
536 IF ARRAY$(I*30-29,I*30)<>BLANK$ THEN PRINT I,ARRAY$(I*30-29,I*30)
537 NEXT I
538 PRINT
539 PRINT "** END OF LIST **"
540 PRINT
541 PRINT "DO YOU WANT TO PRINT THE CUSTOMER LIST TO THE PRINTER (Y/N) ";
542 INPUT YN$
545 IF YN$="N" THEN 640
547 IF YN$<>"Y" THEN 540
550 LPRINT "NUMBER","CUSTOMER NAME"
560 LPRINT
570 FOR I=1 TO 100
580 IF ARRAY$(I*30-29,I*30)<>BLANK$ THEN LPRINT I,ARRAY$(I*30-29,I*30)
590 NEXT I
600 LPRINT
610 LPRINT "* END OF LIST *"
620 LPRINT
630 LPRINT
640 END
```

EMULATING
DOUBLE-SUBSCRIPTED STRING ARRAYS

ROW \ COL	1	2	3	4
1	x x x 1 2 3	x x x 4 5 6	x x x 7 8 9	x x x 10 11 12
2	x x x 13 14 15	x x x 16 17 18	x x x 19 20 21	x x x 22 23 24
3	x x x 25 26 27	x x x 28 29 30	x x x 31 32 33	x x x 34 35 36

A$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx""

1,2,3... ...35,36

To find the starting location of a particular substring in the array A$, use the following formula:

((COL-1)*CHAR)+((ROW-1)*CHAR*NUMCOL)+1

ROW = Row number
COL = Column number
CHAR = number of characters per element of array
NUMCOL = total number of columns

In the example given, CHAR = 3, and NUMCOL = 4. In order to find the starting location of A$(ROW,COL), where ROW=2 and COL=4, perform the following calculation:

((4-1)*3)+((2-1)*3*4)+1 = ((3)*3)+((1)*3*4)+1 = (9)+(12)+1 = 22

Thus, the starting character of substring A$(2,4) is character number 22. This substring is addressed as A$(22,24).

FORMATTING IN ATARI BASIC

Atari BASIC handles formatting in a manner which may be unfamiliar to users of Microsoft BASIC. The two most common Microsoft formatting functions, TAB and SPC, are not present. There are several different approaches which may be used. For screen formatting, the POSITION statement is sufficient. A word can be placed anywhere on the screen simply by specifying the X and Y coordinates. For spacing on the same line, as with the TAB function, the row specification remains the same and the column changes.

A second formatting device is the comma. A comma in a PRINT statement causes the next printed statement to appear in the next print zone. This print zone defaults to 10 spaces, but it can be changed with a POKE statement. If you POKE 201,X the new print zone will be X spaces. However, the easiest way to imitate the TAB function in Atari BASIC is with a string of programmed cursor movements. Include this line in your program:

```
10 DIM R$(80):R$="  [80 cursor overs]   "
```

To program cursor overs, type QUOTE, ESC, CTRL *, ESC, CTRL *, ESC CTRL *, and so on. The cursor over will appear on the screen as a right-arrow character. When R$ is printed, no character will appear, but the cursor will move over 80 spaces.

When you come to a PRINT TAB(X); in your program, replace it with PRINT R$(1,X);. This will print X number of spaces, just as the TAB or SPACE function would. Follow the statement with the word or variable to be printed, just as you would with the TAB or SPC statement.

This method of formatting can be used on either screen or printer. The printer, however, does not perform cursor movements, so R$ should consist of 80 blank spaces. If spaces are used on the screen, they will be printed over any characters which are already there, erasing them.

Here is a sample program using this method of formatting on the printer:

```
10 DIM A$(35),R$(80)
20 R$="                                                "
30 OPEN #1,8,0,"P:"
40 A$="THIS LINE IS INDENTED BY 10 SPACES"
50 PRINT #1; R$(1,10); A$
60 CLOSE #1
```

This program opens a file on the printer and prints to that file. This method is better than using the LPRINT command, because of a limitation in LPRINT which causes extra spaces to be printed where an LPRINT line ends with a semi-colon. This problem can be avoided altogether by opening a file to the printer and printing to that file, as shown here.

TAB STOPS

1. Fixed Width Tab Stops

Memory location 201 controls the print tab width when using the PRINT command with a comma as a field seperator.

```
100 POKE 82,0
110 PRINT PEEK(201)
120 PRINT "A","B","C","D","E"
130 POKE 201,8
140 PRINT "A","B","C","D","E"
```

2. Variable Width Tab Stops

	KEYBOARD SEQUENCES	FUNCTION
▷	ESC TAB	tab to next tab stop
←	ESC CTRL & TAB	clear tab stop
→	ESC CTRL & SHIFT	set tab stop

Tab stops are implemented through special function keys documented on pp. 5-6 of the BASIC Reference Manual. Tab stops can be specified for output to the screen. There are default settings at columns 7,15,23,31,39.

```
150 PRINT "A▷B▷C▷D▷E▷F"
```

To clear a tab stop it is first necessary to tab to the column. The following line will clear the 5 default settings.

```
160 PRINT "▷←▷←▷←▷←▷←"
```

To set a tab stop it is first necessary to space over to the column. The following line will set tabs at columns 5,12,20.

```
170 PRINT "    →       →       →"
180 PRINT "A▷B▷C▷D"
```

Data Files on Cassette

There is a peculiarity of the cassette file handler which could cause problems if you are not aware of it. When a file is OPENed for output on the cassette, the motor and audio channels are turned on, and blank leader is written out to the tape. The motor will stay on and leader will continue to be written out until a 128 byte block of data is output to the file with a PUT or PRINT # statement. If you attempt to turn off the motor by writing to PACTL (POKE 54018,60) then garbage will be written on the tape before it stops, leaving a bad tape. The best solution to this problem is to write out a dummy record to the tape. The length of a cassette record is 128 bytes. Data is accumulated in a buffer until it reaches this length. If you set a dummy variable equal to 128 spaces, and PRINT to the file, the motor will shut off correctly. It will then wait until the next 128 bytes have been accumulated before turning on and dumping the data to the tape. Here is an example of this technique:

```
10 DIM A$(128)
20 FOR I=1 TO 128
30 A$(LEN(A$)+1)=" "
40 NEXT I
50 OPEN #1,8,0,"C:"
60 PRINT #1; A$
70 PRINT #1; "DATA"
80 CLOSE #1
```

Remember, if you have written a dummy record, you must read this record back and throw it away when you INPUT your data.

If you have 128 bytes or more of data which is to be written as soon as the OPEN statement is executed, it will be unnecessary to write out a dummy buffer.

DATA SEPARATORS

When writing data to a file with the PRINT # statement, each piece of data must be separated from the next, so that the INPUT # statement will know where one record ends and another begins. If a separate PRINT statement is used for each variable or piece of data, then each record will automatically be separated by an End-of-Line character. This character, called an EOL, is ATASCI character 155, written CHR$(155). An EOL is written to the tape or disk after each PRINT # statement.

If you wish to put more than one variable or piece of data on the same line, you must supply your own separators. The separator (also called a delimiter) will be an EOL for strings or string variables, or a quoted comma for numbers or numeric variables. In the following example, different combinations of variables are shown. Notice that the PRINT # statement itself is always followed by a semicolon, never a comma. If a comma were used following a PRINT # statement, 10 blank spaces would be written out to the tape, just as they would be to the screen. The example shows a cassette file, but the same PRINT and INPUT statements would be used for disk data files.

```
open file for output:    10 OPEN #1,8,0,"C:"
write string variables:  20 PRINT #1; A$;CHR$(155);B$;CHR$(155);C$
numeric variables:       30 PRINT #1; A;",";B;",";C
quoted strings:          40 PRINT #1; "CAT";CHR$(155);"DOG"
mixed variables and
            constants:   50 PRINT #1; A$;CHR$(155);28;",";B;CHR$(155);B$
                         60 CLOSE #1
```

Notice that when mixed variable types are used, an EOL must separate a string from a numeric variable.

```
1 REM CASSETTE FILE DEMO
2 REM BY PAM YOCUM, ATARI - 3/14/81
3 REM
4 REM THIS PROGRAM CREATES A SIMPLE CASSETTE FILE OF NAMES
5 REM AND ADDRESSES. IT THEN READS THEM BACK IN AND OUTPUTS
6 REM THEM TO THE DISPLAY AND PRINTER.L.
7 REM
9 DIM B$(128),N$(20),A$(40),C$(40),ANS$(1)
10 REM
11 REM FIRST WE OPEN THE CASSETTE DECK FOR OUTPUT
12 REM
13 OPEN #1,8,0,"C:"
15 REM
16 REM NEXT WE MUST SEND OUT A DUMMY RECORD.
17 REM THIS WILL STOP THE RECORDER MOTOR
18 REM
20 FOR I=1 TO 128
30 PRINT #1;" ";
40 NEXT I
45 PRINT #1
47 REM
48 REM NOW TO GET THE DATA
49 REM
50 PRINT
60 PRINT "INPUT NAME";
70 INPUT N$
80 PRINT "INPUT STREET ADDRESS";
90 INPUT A$
100 PRINT "INPUT CITY,STATE";
110 INPUT C$
112 REM
113 REM SEND THE DATA TO THE CASSETTE DECK.
114 REM THE MOTOR WILL TURN ON AS NEEDED.
115 REM
120 PRINT #1;N$
130 PRINT #1;A$
140 PRINT #1;C$
150 PRINT "INPUT Y/N FOR ADDITIONAL INFO";
160 INPUT ANS$
170 IF ANS$="Y" THEN 50
180 IF ANS$<>"N" THEN 150
182 REM
183 REM THE 'CLOSE' COMMAND WILL WRITE OUT
184 REM THE REMAINDER OF THE DATA
185 REM
190 CLOSE #1
191 REM
192 REM NOW WE CAN READ THE DATA BACK IN
193 REM
194 PRINT "REWIND RECORDER AND SET IT TO 'PLAY'"
195 REM
196 REM NOW OPEN THE RECORDER FOR INPUT
197 REM
198 OPEN #1,4,0,"C:"
200 REM
201 REM DELETE LINES 204,262,264,266,268 IF YOU DON'T
202 REM HAVE A PRINTER
203 REM
204 OPEN #2,8,0,"P:"
205 TRAP 280
206 REM THIS GETS RID OF THE DUMMY RECORD
207 INPUT #1,B$
208 REM NOW LET'S GET THE DATA
```

```
210 INPUT #1;N$
220 INPUT #1;A$
230 INPUT #1;C$
235 REM DISPLAY THE DATA ON THE SCREEN
240 PRINT N$
250 PRINT A$
260 PRINT C$
261 REM THESE LINES PRINT ON THE PRINTER
262 PRINT #2;N$
264 PRINT #2;A$
266 PRINT #2;C$
268 PRINT #2
270 GOTO 210
280 CLOSE #1
290 END
```

```
10 REM 'D:BIRTHDAY' WEB 6-19-81
100 REM MAIN LINE
110 DIM MONTH$(16),FILE$(16),NAME$(30),DATE$(8),IO$(1)
120 GRAPHICS 0
130 PRINT "BIRTHDAY FILE PROGRAM"
200 REM DEFINE MONTH
210 TRAP 200
220 FILE$="D:BIRTHDAY."
230 PRINT :PRINT "MONTH (1-12)(0=END)";:INPUT MON
240 IF MON=0 THEN 900
250 IF MON<1 OR MON>12 THEN 200
260 GOSUB 1000+MON
270 FILE$(12,14)=MONTH$
300 REM TRY TO OPEN FILE.
310 TRAP 500
320 MODE=8
330 OPEN #1,4,0,FILE$
340 PRINT :PRINT "BIRTHDAYS IN ";MONTH$
350 TRAP 400
360 INPUT #1,NAME$,DATE$
370 PRINT NAME$,DATE$
380 GOTO 360
400 REM END OF CURRENT FILE
410 CLOSE #1
420 MODE=9
430 GOTO 600
500 REM MONTH FILE NOT YET CREATED
510 PRINT :PRINT "NO BIRTHDAYS DEFINED FOR ";MONTH$
520 MODE=8
530 CLOSE #1
600 REM ALLOW ADDING TO FILE
610 PRINT :PRINT "DO YOU WISH TO ADD NEW DATA (Y/N)";:INPUT IO$
620 IF IO$<>"Y" THEN 200
630 OPEN #1,MODE,0,FILE$
640 IO$=CHR$(155)
650 PRINT :PRINT "TYPE IN NAMES AND DATES"
660 PRINT :PRINT "NAME";:INPUT NAME$:IF LEN(NAME$)=0 THEN 700
670 PRINT "DATE";:INPUT DATE$
680 PRINT #1;NAME$;IO$;DATE$
690 GOTO 660
700 REM END OF ADDS
710 CLOSE #1
720 GOTO 200
900 REM END
910 GRAPHICS 0
920 END
1001 MONTH$="JANUARY":RETURN
1002 MONTH$="FEBRUARY":RETURN
1003 MONTH$="MARCH":RETURN
1004 MONTH$="APRIL":RETURN
1005 MONTH$="MAY":RETURN
1006 MONTH$="JUNE":RETURN
1007 MONTH$="JULY":RETURN
1008 MONTH$="AUGUST":RETURN
1009 MONTH$="SEPTEMBER":RETURN
1010 MONTH$="OCTOBER":RETURN
1011 MONTH$="NOVEMBER":RETURN
1012 MONTH$="DECEMBER":RETURN
```

```
1 REM **DEMO:OF ALTERNATIVE USE OF
2 REM .  CODE 9 (P. 27-BASIC REF MAN)
3 REM .  BY PAM YOCUM, ATARI, 10/22/80
4 REM
10 DIM A$(1000),B$(40)
15 A$=""
16 PRINT "}USE SCREEN EDITOR KEYS TO TYP
E"
17 PRINT "CHARACTERS ON SCREEN, SCATTERI
NG "
18 PRINT "THEM AROUND, SOME ON EACH LINE
.":PRINT :PRINT "HIT RETURN WHEN READY";

19 INPUT B$
20 OPEN #1,9,0,"E:"
24 REM
25 REM **ALLOW USER TO TYPE THINGS
30 INPUT B$
39 REM
40 REM **READ SCREEN LINE BY LINE
60 FOR Y=0 TO 23
70 POSITION 2,Y
74 REM
75 REM **FOLLOWING LINE RETURNS
76 REM .  EVERYTHING FROM CURRENT
77 REM .  CURSOR POSITION TO END OF
78 REM .  LINE.
80 INPUT #1;B$
84 REM
85 REM **CONCAT. TO A$ TO SAVE
86 REM .  ENTIRE SCREEN LINE BY LINE
90 A$(LEN(A$)+1)=B$
95 A$(LEN(A$)+1)=CHR$(155):REM EOL
100 NEXT Y
104 REM
105 REM ***PRINTING BACK THE ENTIRE
106 REM .  SCREEN. AT THIS POINT
107 REM .  COULD SAVE TO DEVICE, ETC.
120 PRINT "}HIT RETURN TO DISPLAY SCREEN
";
122 INPUT B$
130 PRINT A$(1,LEN(A$)-1);
135 REM
136 REM **<BREAK> TO EXIT PROGRAM
140 GOTO 140
```

```
100 REM . BUBBLE-SORT
110 REM . BY PAM YOCUM - ATARI - 11/80
120 REM
130 DIM A(100)
140 REM
150 REM . READING NUMBER OF ITEMS TO BE
160 REM . SORTED & DATA TO BE SORTED
170 REM . INTO AN ARRAY
180 REM . ****************************
190 READ N
200 FOR B=1 TO N
210 READ X:A(B)=X
220 NEXT B
230 REM
240 REM . PRINTING UNSORTED LIST
250 REM . ***********************
260 PRINT "UNSORTED LIST"
270 FOR B=1 TO N
280 PRINT A(B)
290 NEXT B
300 REM
310 REM . THE SORT
320 REM . ********
330 F=0:REM ***FLAG=1 IF NEEDS MORE SORTING
340 FOR C=1 TO N-1
350 IF A(C+1)>=A(C) THEN 410
360 T=A(C):REM ***TEMPORARY REGISTER
380 A(C)=A(C+1)
390 A(C+1)=T
400 F=1
410 NEXT C
420 IF F=1 THEN 330
430 REM
440 REM . PRINTING SORTED LIST
450 REM . ********************
460 PRINT
470 PRINT "SORTED LIST"
480 FOR B=1 TO N
490 PRINT A(B)
500 NEXT B
510 REM
520 REM . DATA ITEMS
530 REM . **********
540 DATA 10
550 DATA 9,1,5,10,4,7,2,8,3,6
560 END
```

```
10 REM 'ROCKSORT'    WBB 4-10-81
100 REM MAIN LINE
110 DIM SORT$(80),BUF$(1),TONE(80)
120 GRAPHICS 0:POKE 82,0
125 FOR I=1 TO 80 STEP 2:READ T:TONE(I)=T:TONE(I+1)=T:NEXT I
130 PRINT "80 CHARACTER STRING TO BE SORTED..."
140 INPUT SORT$
150 LAST=LEN(SORT$)
160 GRAPHICS 2
170 PRINT SORT$
180 POKE 752,1
190 POKE 18,0:POKE 19,0:POKE 20,0
200 FOR I=LAST-1 TO 1 STEP -1
205 SOUND 0,TONE(I),10,10
210 K=0
220 FOR J=1 TO I
225 SOUND 1,TONE(J),10,8
230 IF SORT$(J,J)>SORT$(J+1,J+1) THEN BUF$=SORT$(J,J):SORT$(J,J)=SORT$(J+1,J+1):
SORT$(J+1,J+1)=BUF$:K=1
240 POSITION 0,1
250 PRINT #6;SORT$
260 NEXT J
265 IF K=0 THEN I=1
270 NEXT I
280 POKE 752,0
290 SOUND 1,0,0,0
300 FOR I=29 TO 0 STEP -1:SOUND 0,I,10,10:NEXT I
310 GOSUB 9900
320 PRINT "ELAPSED TIME ";HH;":";MM;":";SS;
330 END
1000 DATA 29,31,33,35,37,40,42,45,47,50,53,57,60,64,68,72,76,81,85,91,96,102,106
,114,121,128,136,144,153
1001 DATA 162,173,182,193,204,217,217,230,230,243,243
9900 REM 'TIME.SUB' WBB 3-12-81
9902 REM RETRIEVES REAL TIME CLOCK
9904 REM (OUT-HH,MM,SS)
9906 SS=(PEEK(18)*256*256+PEEK(19)*256+PEEK(20))/60
9908 HH=INT(SS/(60*60))
9910 SS=SS-HH*60*60
9912 MM=INT(SS/60)
9914 SS=INT(SS-MM*60)
9916 RETURN
```

```
1000 REM .  PROGRAM ETCH-A-SKETCH
1005 REM .  BY PAM YOCUM, 9/4/80
1010 REM
1020 REM .  THIS PROGRAM SIMULATES AN
1030 REM .  ETCH-A-SKETCH USING A JOY-
1040 REM .  STICK TO DRAW THE LINE
1050 REM
1060 REM .  VARIABLES
1070 REM .  *********
1080 REM .    XSTEP( )..ARRAY FOR X-AXIS
1090 REM .              INCREMENTS
1100 REM .    YSTEP( )..ARRAY FOR Y-AXIS
1110 REM .              INCREMENTS
1120 REM .    X,Y.....X & Y COORDIN.
1130 REM .    I,J.....INDEX VARIABLES
1135 REM .    S.......VALUE RETURNED
1136 REM .              BY STICK(0)
1140 REM
1150 REM .  SET UP INITIAL CONDITIONS
1160 REM .  ************************
1170 DIM XSTEP(20),YSTEP(20)
1180 SETCOLOR 0,2,8
1190 COLOR 1
1200 GRAPHICS (8+16)
1210 X=160:REM ***STARTING X LOCATION
1220 Y=91:REM ***STARTING Y LOCATION
1230 REM
1240 REM .  SETTING UP ARRAYS FOR X & Y
1250 REM .  INCREMENTS. SUBSCRIPTS REFER

1255 REM .  TO JOYSTICK POSITIONS.
1260 REM .  ***************************
1270 XSTEP(5)=1:YSTEP(5)=1
1280 XSTEP(6)=1:YSTEP(6)=-1
1290 XSTEP(7)=1:YSTEP(7)=0
1300 XSTEP(9)=-1:YSTEP(9)=1
1310 XSTEP(10)=-1:YSTEP(10)=-1
1320 XSTEP(11)=-1:YSTEP(11)=0
1330 XSTEP(13)=0:YSTEP(13)=1
1340 XSTEP(14)=0:YSTEP(14)=-1
1350 XSTEP(15)=0:YSTEP(15)=0
1355 REM
1356 REM .  TO TURN WARNING BEEP OFF
1357 REM .  ************************
1360 SOUND 0,0,0,0
```

```
1361 REM
1363 REM .  TESTING TRIGGER BUTTON. IF
1364 REM .  BUTTON PUSHED ERASES SCREEN
1365 REM .  TO START OVER.
1366 REM .  ***************************
1370 IF STRIG(0)=0 THEN 1180
1380 S=STICK(0)
1390 REM
1400 REM . DETERMINING INCREMENTS FOR X
1410 REM . & Y AND PLOTTING NEW POINTS
1420 REM . ***************************
1430 X=X+XSTEP(S):Y=Y+YSTEP(S)
1440 TRAP 1600:REM ***CHECKING OUT OF
1450 REM .              BOUNDS ERROR
1460 PLOT X,Y
1470 GOTO 1370
1480 REM
1490 REM . THE FOLLOWING LINES ARE
1500 REM . EXECUTED IF TRY TO MOVE OUT
1510 REM . BOUNDS WHEN DRAWING ON THE
1520 REM . SCREEN. THE LAST POINT IS
1530 REM . MOVED BACK IN BOUNDS AND THE
1540 REM . WARNING SOUND IS ACTIVATED.
1550 REM . THE FOR-NEXT LOOPS ARE USED
1560 REM . TO DETERMINE LENGTH OF WARN-
1570 REM . ING BEEP (J) AND CAUSES IT
1580 REM . TO FADE IN INTENSITY (I)
1590 REM . ***************************
1600 X=X-XSTEP(S):Y=Y-YSTEP(S)
1620 FOR I=15 TO 0 STEP -1
1630 FOR J=1 TO 6
1635 NEXT J
1640 SOUND 0,136,10,I
1650 NEXT I
1660 GOTO 1360
1670 END
```

```
1000 REM .  PROGRAM MOVE-A-BOX
1005 REM .  BY PAM YOCUM, 9/5/80
1010 REM
1020 REM .  THIS PROGRAM MODIFIES THE
1030 REM .  ETCH-A-SKETCH PROGRAM SO
1040 REM .  THAT THE JOYSTICK IS USED
1045 REM .  TO MOVE AROUND A BOX.
1050 REM
1060 REM .  VARIABLES
1070 REM .  *********
1080 REM .    XSTEP( )..ARRAY FOR X-AXIS
1090 REM .              INCREMENTS
1100 REM .    YSTEP( )..ARRAY FOR Y-AXIS
1110 REM .              INCREMENTS
1120 REM .    X,Y......X & Y COORDIN.
1130 REM .    I,J......INDEX VARIABLES
1140 REM .    S........VALUE RETURNED
1150 REM .            BY STICK(0)
1160 REM
1170 REM .  SET UP INITIAL CONDITIONS
1180 REM .  ************************
1190 DIM XSTEP(20),YSTEP(20)
1200 SETCOLOR 0,2,8
1210 COLOR 1
1220 GRAPHICS (7+16)
1230 X=80:REM ***STARTING X LOCATION
1240 Y=40:REM ***STARTING Y LOCATION
1250 REM
1260 REM .  SETTING UP ARRAYS FOR X & Y
1270 REM .  INCREMENTS. SUBSCRIPS
1275 REM .  REFER TO JOYSTICK POSITIONS
1280 REM .  ***************************
1290 XSTEP(5)=1:YSTEP(5)=1
1300 XSTEP(6)=1:YSTEP(6)=-1
1310 XSTEP(7)=1:YSTEP(7)=0
1320 XSTEP(9)=-1:YSTEP(9)=1
1330 XSTEP(10)=-1:YSTEP(10)=-1
1340 XSTEP(11)=-1:YSTEP(11)=0
1350 XSTEP(13)=0:YSTEP(13)=1
1360 XSTEP(14)=0:YSTEP(14)=-1
1370 XSTEP(15)=0:YSTEP(15)=0
1375 GOTO 1580
1380 REM
1390 REM .  TO TURN WARNING BEEP OFF
1400 REM .  ************************
1410 SOUND 0,0,0,0
1420 REM
1430 REM .  TESTING TRIGGER BUTTON. IF
1440 REM .  BUTTON PUSHED STARTS OVER.
1460 REM .  ******************************
1470 IF STRIG(0)=0 THEN 1200
1480 S=STICK(0)
```

```
1482 REM .  FOLLOWING CAUSES A PAUSE SO
1484 REM .  BOX STAYS ON SCREEN WHILE
1486 REM .  WAITING TO MOVE IT
1488 REM .  ***************************
1490 IF S=15 THEN 1470
1500 REM . ERASING BOX BY SETTING COLOR
1510 REM . TO BACKGRD COLOR AND DRAWING
1520 REM . BOX IN BACKGROUND COLOR,
1525 REM . THEN SET BACK TO BOX COLOR
1530 REM . ***************************
1540 COLOR 0:REM ***TRY TAKING OUT THIS
1544 REM .            LINE AND LINE 1880
1545 REM .            TO SEE HOW BOX IS
1546 REM .            NOT ERASED.
1550 GOSUB 1990:REM ***TO DRAW BOX
1580 COLOR 1
1590 REM
1600 REM . DETERMINING INCREMENTS FOR X
1610 REM . & Y AND PLOTTING NEW POINTS
1612 REM . MULTIPLIES BY 5 TO SPEED UP
1614 REM . MOVEMENT OF BOX
1616 REM .***************************
1620 X=X+XSTEP(S)*5:Y=Y+YSTEP(S)*5
1630 TRAP 1870:REM ***CHECKING OUT OF
1640 REM .             BOUNDS ERROR
1650 PLOT X,Y
1660 GOSUB 1990:REM ***TO DRAW BOX
1670 GOTO 1470
1760 REM . THE FOLLOWING LINES ARE
1770 REM . EXECUTED IF TRY TO MOVE OUT
1780 REM . BOUNDS WHEN DRAWING ON THE
1790 REM . SCREEN. THE LAST POINT IS
1800 REM . MOVED BACK IN BOUNDS AND THE
1810 REM . WARNING SOUND IS ACTIVATED.
1820 REM . THE FOR-NEXT LOOPS ARE USED
1830 REM . TO DETERMINE LENGTH OF WARN-
1840 REM . ING BEEP (J) AND CAUSES IT
1850 REM . TO FADE IN INTENSITY (I)
1860 REM . ***************************
1870 X=X-XSTEP(S)*5:Y=Y-YSTEP(S)*5
1880 GRAPHICS (7+16):COLOR 1
1900 FOR I=15 TO 0 STEP -1
1910 FOR J=1 TO 3
1920 NEXT J
1930 SOUND 0,136,10,I
1940 NEXT I
1950 GOTO 1630
1955 STOP
1960 REM
1970 REM . SUBROUTINE FOR DRAWING BOX
1980 REM . ***************************
1990 DRAWTO X+10,Y
2000 DRAWTO X+10,Y+10
2010 DRAWTO X,Y+10
2020 DRAWTO X,Y
2030 RETURN
2040 END
```

```
100 REM THIS DEMO SHOWS HOW TO USE
101 REM LOWER CASE IN GR. 1
102 REM WITHOUT FILLING THE SCREEN
103 REM WITH HEARTS.
104 REM THE DATA IS TAKEN FROM THE
105 REM CHARTS ON P 55-56 OF
106 REM THE BASIC REFERENCE MANUAL
110 REM ********************************
200 GRAPHICS 1:N=1
205 POKE 756,226:SETCOLOR 0,0,0
210 READ X:IF X=999 THEN GOTO 300
215 N=N+1
220 COLOR X+64:PLOT 1,N:DRAWTO N,N
230 GOTO 210
240 DATA 42,53,36,57,0,34,47,39,33,50,52
,999
300 GOTO 300
```

```
10 REM NO MORE HEARTS
11 REM IN LOWER CASE, GRAPHIC 1 & 2
12 REM ****************************
15 IF FRE(0)<4600 THEN END
20 RAMTOP=PEEK(106)
25 POKE 106,RAMTOP-16
30 GRAPHICS 1
40 CHBAS=RAMTOP-4:ADDR=CHBAS*256
50 FOR I=0 TO 1023
60 POKE ADDR+I,PEEK(57344+I)
70 NEXT I
80 CHAR=64:POS=ADDR+(CHAR*8)
90 DATA 0,0,0,0,0,0,0,0
100 FOR X=0 TO 7
110 READ A
120 POKE (POS+X),A
130 NEXT X
150 REM ****************************
160 REM USE THIS ROUTINE TO
170 REM INITIALIZE YOUR PROGRAM
180 REM CALL LOWER CASE WITH
190 REM POKE 756, CHBAS+2
195 REM ****************************
200 POKE 756,CHBAS+2
210 POSITION 5,5
220 PRINT #6;"LOWER CASE"
230 POSITION 3,8
240 PRINT #6;"WITHOUT HEARTS"
```

GRAPHICS
Mode 0
Charac. in
Mode 8

ATARI®

inter-office memo

DATE: 9/26/80

SUBJECT: Mode 0 Characters in Graphics Mode 8

This example will show how to display text using the standard 128 character set on a graphics mode 8 screen. The characters may be positioned horizontally in the standard 40 columns of mode 0. The vertical position however may be on any of the 192 mode 8 lines.

To define the position of the character on the screen set the variable X to the horizontal position (0-39), and set Y to the vertical position (0-191). Use the example to output characters.

```
 10   DIM A$(4),0$(1)
 20   A$="TEST"
 30   X=15:Y=80
 40   GRAPHICS 8
 50   I1=PEEK(88)+PEEK(89)*256
 60   I2=I1+Y*40+X
 70   FOR Z=1 TO LEN(A$)
 80   0$=A$(Z,Z)
 90   GOSUB 200
100   I3=57344+X*8
110   FOR U=0 TO 7
120   POKE I2+U*40,PEEK(I3+U)
130   NEXT U
140   I2=I2+1
150   NEXT Z
160   STOP
200   X=ASC(0$)
210   IF X>127 THEN X=X-128
220   IF X>31 AND X<96 THEN X=X-32:RETURN
230   IF X<32 THEN X=X+64
240   RETURN
```

```
5 REM FILL IN A SHAPE
6 REM BY LANE WINNER 4/81
10 GRAPHICS 7+16:COLOR 1:DEG
20 GOSUB 500:PLOT X,Y
30 FOR T=0 TO 360 STEP 6
40 GOSUB 500:DRAWTO X,Y
50 NEXT T
80 X=79:Y=47
100 FILL=1000:TRAP 2000
200 GOSUB FILL
300 GOTO 300
500 Y=30*COS(T)+47+10*COS(T*2)
510 X=30*SIN(T)+79+10*SIN(T*3)
520 RETURN
1000 PLOT X,Y
1010 X=X-1:LOCATE X,Y,Z
1020 IF Z=0 THEN GOSUB FILL
1030 X=X+1
1040 Y=Y-1:LOCATE X,Y,Z
1050 IF Z=0 THEN GOSUB FILL
1060 Y=Y+1
1070 X=X+1:LOCATE X,Y,Z
1080 IF Z=0 THEN GOSUB FILL
1090 X=X-1
1100 Y=Y+1:LOCATE X,Y,Z
1110 IF Z=0 THEN GOSUB FILL
1120 Y=Y-1
1130 RETURN
2000 TRAP 2000:GOTO PEEK(186)+PEEK(187)*256+20
```

PSAN 1-89

Comparing The New Graphics Operating Desktops for the 8-Bit Atari:

DIAMOND OS ST jr

(by Alan Reeve of REEVE Software, distributed by USA Media)

and GOE (Graphics Operating Environment)

(by David R. Sullivan of Total Control Systems)

Reviewed by Penny Ormston, R-Atari Club

CONFUSION

Ok, right off there are probably a few people reading this who are scratching their heads over the above information. "What?", they are probably thinking, "Aren't the STjr and the GOE the same program?" The answer is NO! Understandably there is a lot of confusion going on about these two similar products. Not only are they similar in appearance and in what they do, but apparantly even the magazines are having difficulty in telling them apart. In the October 1988 issue of Analog, for example, there is quite a write up of the GOE STjr available from Merrill Ward (now known as USA Media). This write up really threw me for a loop, because when I attended the Atari Faire in Glendale, CA in mid September, I saw BOTH the STjr and the GOE — at two seperate booths, and quite obviously... NOT one and the same product!

Part of the confusion lies in the fact that USA Media's press release for the STjr has several refrences to itself as being a GOE or Graphic Operating Environment. In fact, the press release contains so many names, it is difficult to tell which one is actually the name of their product other than STjr. They alternately call it "St. Jr. Graphics Operating Environment", "Graphic Operating Environment ST. Jr.", "Diamond GOE ST. Jr.", and simply "ST. Jr.". No wonder the Analog writer didn't get it straight, the USA Media people can't seem to figure out what it is called either! The outside of the box calls it the Diamond OS STjr, and for here on out when I refer to it I will simply call it "STjr". (whew!) The STjr has been released in disk form for the price of $29.95. A cartrige version is in the works and has not been released yet, but is expected to cost $49.95.

In the meantime, GOE, which is written by David R. Sullivan and distributed by Total Control Systems, has only been released in its DEMO form. The actual cartrige has not been released yet. The GOE cartrige costs $50.00. The GOE demo is made readily available to all interested persons and users groups free of charge, so you can get a look at it before you buy. I certainly wish more companies would do this, I really think it would curb a lot of pirating if people were able to get a free peek at what's available and decide from that whether they want to buy or not.

Another reason for the apparant confusion between the GOE and the STjr stems from the fact that David Sullivan (GOE) used to work for USA Media (STjr). According to Mr. Sullivan, in his "Special Notice" text file on the GOE-demo disk, at one point Total Control Systems and Merrill Ward and Assoc. were working on an agreement to publish GOE. The press release was made before an agreement could be reached. Mr. Sullivan goes on to make some rather strong allegations against Merrill Ward and Assoc., now known as USA Media, including the following:

"Merrill Ward & Assoc. now under the name of USA Media is associated with another product. The product that they currently sell is NOT GOE. Although it has been advertised as such. They persist to use GOE, Graphics Operating Environment in such a way to mislead you, the public, into believing that their product is GOE. It is not, Total Control Systems is the ONLY publisher of GOE at the present time."

Later in the same text file, David Sullivan makes this statement: "I can only stress to the Public, PLEASE USE EXTREME CAUTION when reading press from these people now or in the future."

In speaking to Mr. Sullivan on the phone, I can really feel the animosity he feels toward USA Media. He feels that they are not above stealing his code, and he stipulates in his demo's text files that he does not want a copy of it to slip into the hands of Shelly Merrill. He also stated that he does NOT feel this way about Alan Reeve, the writer of STjr.

I have tried to reach both Alan Reeve and Shelly Merrill to ask questions about the STjr and even about David Sullivan's accusations, but have not been able to reach either of them at this time. This is particularly irritating because it costs me money to try to call, plus I can't get any answers to my questions about the STjr. I think it is very important for users to be able to contact the software publishers when they have problems or questions. We need the support even after we have spent our money on the products. Apparently Mr. Merrill is in the process of moving, and that is the reason I was unable to reach him. I can understand that situation. But it took two phone calls just to find out that much, and my questions have remained unanswered.

I hope this clears up a little of the confusion about these two products instead of adding to it. Obviously we have two highly competitive products here, now I will try to sort out the strengths and weaknesses of both products. Please note, that I only have a free demo of the GOE to go on, not the actual cartrigel I ordered one at the Atari Faire, but at the time of this writing it has not been released. The STjr that sits in front of me is the $29.95 disk version. This may seem like an unfair comparison, made between a demo and a full-fledged product, but I think it is important to set the record straight about these products. When the cartrige versions of these two products are released, they will be re-reviewed in depth. For now, this is just an overview/ comparison.

WHAT IS A DESKTOP?

ST users are well aquainted with desktop programs, such as GEM, but for most of us who are strictly 8 bitters the concept may be a little vague.

What is a Desktop? Well, to put it as simply as possible, it is a graphic Disk Utility Package similar to what you normally use with DOS. The difference is that the Desktop uses icons (pictures) instead of words, and you can use the joystick, mouse, or keyboard arrow keys to make your selections. This is supposed to make it easier, because you do not have to memorize commands such as A, Dir, Format, I, etc. to check a directory, format a disk, or whatever. Both the GOE and the STjr can use all three input devices. Both look much nicer that a typical DOS screen.

Is that all that either of these products is, a graphic DOS? NO! Not at all. Both the GOE and STjr have

ditional support program in the works, ord processors, paint programs, and ore. All supporting the same Graphic vironment as the parent product. Now r the comparisions:

APPEARANCE:

Both the STjr and the GOE are milar in appearance. The STjr shows a reen screen with a file icon and trashcan con on the left side of the screen, and our menu words across the top of the creen: Desk File Disk Options. If the ystem is booted with more than one rive online, there is still only one file con shown. If the file icon is selected, ou can "double click" to open the file the icon drawer opens to show you have ccessed the drive). Now you can check he files on the disk or whatever.

GOE shows a green screen with two disk icons (labeled "A" and "B") and a trashcan icon. It shows two disk icons no matter how many drives are present at logon. The GOE has four menu words across the top of the screen also: Desk File View Option.

INPUT DEVICES

Both the STjr and GOE allow the use of a joystick, mouse, or the keyboard for input. One convience of the STjr over the GOE is being able to use the arrow keys to move the cursor without having to hold the control key as well. However, the GOE allows keyboard input for information as well as cursor movement, so it seems necessary to have the control-arrow combination for movement. Still, it would be nice if there were an option to select keyboard movement using arrow keys alone or with control for different uses.

DOS COMPATIBILITY

The STjr is compatible only with Atari Dos 2.0. The XEversion, demoed on the flip side of the disk, is used with DOSXE or Sparta dos X. The upcoming cartrige version will also be compatible with Dos 2.5.

The GOE demo is compatible with Sparta dos 3.2 and Sparta dos X. A few of the demo functions (such as print to the screen) do not work properly with Sparta Dos X in this demo version. I talked to Mr. Sullivan about this, and he said it was because Sparta Dos X had not been released yet when he made the demo disk. Since Sparta Dos X was first released at the Glendale Arati Faire, I have no doubt that that was exactly the case. The cartrige version will be compatible with Atari dos 2.0, 2.5, Mydos, Smartdos, and Ados as well as the Sparta dos versions.

OPTIONS

STjr:

The four words at the top of the STjr screen: Desk File Disk and Options, are drop down menus to allow you to set up or use the desktop. Moving the arrow up to any of these words will make a menu drop down. Selecting Desk, you will see the option "info". If the arrow is moved to the word info, the word is hilighted. You may now select this, or move off of the menu and do something else. Selecting info will display a title screen, informing you that this is the Diamond OS STjr, etc. Select cancel to do something else.

The file option menu for STjr includes the following options: Open, Status, Duplicate, New Folder, Close, Quit. Unless the file is open, you can't use any of these options except Quit. The option to duplicate is used to copy a file. New Folder is used for subdirectories. Since the regular disk version is only compatible with Atari dos 2.0, this seems like a useless option. Status, I assume, is to show the status of the file...locked, unlocked, deleted? I really don't know. I have never been able to get this function to work. This is one point I wanted to discuss with either Shelly Merril or Alan Reeve, but as I stated before I could not get ahold of either of them. Close closes a file drawer that has been opened.

The Disk option menu has only two choices: Format and Duplicate. These options are obvious.

The Option menu has the following choices: Display, Confirm, Install, Save-Desktop. I assume these options are available to set up the desktop exactly the way you want it, then save your configuration to disk for future use. I was unable to use this. No matter what I did, I just couldn't click on any of the options except the ones that were already marked. I don't know...perhaps my disk is bad. But I was unable to get ahold of anyone to confirm this. The thought that came to my mind though, is that if I have difficulty making this software work, what is a new user going to do?

GOE:

The first of GOE's four options is Desk. Just as in the STjr, GOE's options, when selected, product a drop-down menu. The Desk menu has these options: Desktop Info, Control Panel, Calculator. The Desktop Info will display a title screen letting you know that this is the GOE by David R. Sullivan, and other pertinent information. Just as in the STjr, you click Cancel when you are through looking at the screen so you can get back to working. Selecting the Control panel will show you some marvelous wonders! Unlike the STjr, GOE gives you full control over your desktop. So, you don't like the green screen? You can change it to red, black, or white! Does the arrow move too quickly or slowly for you? You can select the arrow speed for all three input devices right here in the control panel.

Don't have an R time cartridge? No problem. With the Control Panel you can select the time and date yourself. There is an option to change the font density to either 40 or 64 columns. There are other options here as well, but they are either deactivated in the demo, or I just can't figure out what they are for.

My only complaint about the control panel is that once you make a selection the panel disappears. If you want to make any other changes, you must select Desk from the top of the screen again, then go to Control Panel again. It would be so much easier if control would remain with the control panel until you select "cancel" to get back to the main selections. The final option on GOE's desk menu is a calculator. The calculator looks just like a little calculator. You can either use the pointer or the keyboard to input numerals, but the pointer must be used to select operands. The calculator is simple, with no special features, but still it is an attractive, well designed addition to the package.

The file option menu is next in line. Here we have the options: info on disk, New folder (subdirectory), Add drive, Save Desktop, Print Screen, Format Disk. Most of these options are very Basic. Most are also disabled for the demo. The Print Screen option is strange...why would you want to print the screen? I suppose it is like a file dump and the option will be more useful with future additions to the GOE line.

The next is the View menu. This has the following offerings: Icons, Text, Sort by Name, Sort by Type. Using this you can look at your disk files using either Icons or regular text filenames. Using icons looks fancier, but the text seems more practical to me. No matter which you choose, you can read or load any file by selecting it after selecting the disk icon. The sorting is done by name or type, and affects the order in which the files are shown. To see the files, move the pointer-arrow over to a disk icon and give a "double click". If you select drive B and you only have drive A online, then you will see the contents of drive A.

Lastly comes the Option menu. This has the following choices: Execute Dos, Atari BASIC, TOP Cartrige, Missile Com-

and, Reboot System. Missile command is only available n the XEGS. The Atari BASIC and TOP Cartrige ommands are disabled on the demo, but both should be elf explainatory. That leaves us with the two remaining ptions of Execute Dos and Reboot System. I guess we ll know what those options do also, although my son vanted to know why anyone would ever want to kill OS. (Ahem...)

DOCUMENTATION

STjr's documentation is very simple. There just isn't much there. About 20 pages, and nearly half of that is discussion of future Diamond products. The documentation talks much more about what it is, and what it can do, and will do in the future, but says very little about how to do it. Clear documentation would have been a big plus. Instead, there were several functions I just couldn't use or figure out.

The GOE demo has a lot of documentation on the disk. There are six text files on the disk, but only one really explains the use of the GOE demo. This is very basic. Simple and easy to understand. There is one thing that bothered me about it though. Every single page of the text files had several typographical errors or misspellings. I realize that this is only a demo disk, and not the finished product, but seeing so many typos leads me to believe that the person who did this is just throwing it together without taking the time to do it right. If he is sloppy here, maybe he was sloppy with his code?

FUTURE PRODUCTS

Both the STjr and the GOE have support programs "in the works" including word processors and paint programs. There are sample paint programs on both disks, but they are only demos and are not functional. I believe that BOTH of these products will be fully supported by their authors.

SOME CONCLUSIONS

There are many features of both products that I have only touched upon here. As I said before, when the cartrige versions are released, both products will be reviewed again in much greater depth.

Many of the features of GOE are not supported in the demo. This demo is meant to give prospective buyers a look at what is available to them. The final GOE will be user programmable, with full complete (and carefully edited) documentation.

On the other hand, STjr will also release a cartrige soon. But the STjr diskette, which sells for $29.95 is not advertised as being a demo of the upcoming cartrige. If you want to upgrade to the cartrige, you are still out the $29.95 that it cost for the disk.

Personally, I think the free GOE demo is nicer, works better, and does more than the STjr. It is also easier to use, and its author is available to answer questions if you have any problems. These are all BIG PLUSES. It is not perfect, but I have seen enough to believe that Mr. Sullivan will fulfill all my expectations for the GOE cartrige. Unfortunately, I cannot say the same about the Diamond STjr.

DISK DIRECTORY SALVAGE

An 8-Bit "U-Fix-it"!

By Bob Prince

(Reprinted from the NACEC Newsletter 6/88, via ACCOC 10/88)

While cleaning up some files the other day, I had a dreaded experience. A directory call from DOS produced the feared message "ERROR 144". What was on that disk? I didn't know! All I knew was that it was a full double density disk of documentation files. So I booted up my trusty disk utility DISKWIZ 2 (vintage ,1982 from Jerry Allen). Many other sector editors are available to let you look at the contents of a disk and change anything you like.

What I found confirmed my worst fears - two bad sectors, sector 361 and 362. Unless you know something about Atari file structure, the significance of this is not obvious.

After all, what's the big deal about losing two sectors of text? Well what that means in this case is that the first two sectors of the DIRECTORY were lost. Short of programming to read the disk contents into a series of files and editing them into meaningful files, the whole disk was useless. (Not quite, since I did manage to copy five files, but you get my point).

Back to DISKWIZ 2. I mapped the disk and found 26 files on it - all consecutive. Then I went to the first sector of each file and read it to see the name of it. Between what I was able to salvage from my disk, what I had in backups and what I found posted on local BBs's I could replace about half of them. That still left 99K bytes of information lost. I decided to try to salvage it. I checked the file links and was relieved to learn they were intact.

I reasoned that all I had to do was reconstruct an acceptable directory so DOS could find the start of each file and then copy file by file to another disk.

I guess it's time to have a brief discussion on how DOS manages files. The directory is contained on sectors 361 - 368. That's 8 sectors. Each directory sector has 128 bytes and each file takes 16 bytes to record all necessary information:

INFORMATION	BYTE(S)
FILE STATUS	1
FILE LENGTH (sectors)	2,3
FILE START SECTOR	4,5
FILENAME	6-13
EXTENSION	14-16

That allows 8 files per directory sector times 8 directory sectors for a maximum of 64 files per disk. For your information, all data is shown in hexidecimal (ugh!) and byte pairs are always in low byte - high byte order (for example decimal 525 equals hex 020D, but would be entered 0D 02). File status could be never used (00), normal (42), locked (62), deleted (80), or open (43 or 63).

Next I decided to enter the information from my map and manual file scan in the order and format described above. One reason I chose DISKWIZ 2 for this job is that it has some handy features such as a dec-hex conversion, a directory screen print and a graphic screen dump compatible with my printer (Gemini 10X). The formatted directory proved very useful in checking my editing, and before long I had sucessfullly completed my disk surgery...and learned quite a bit about file structure in the process.

STANDARD ATARI DISKETTE FORMAT

BRUCE SMITH

AND IN THE BEGINNING

This is a re-write of the handout passed out at the September 5th general meeting. I have restructured it somewhat, plus filled some gaps and corrected a few errors. This is a better reference document than the handout, and so I would suggest you throw out the handout, and keep this instead. If there is much interest in DOS 3 disks, let me know, and I will do a little research.

Later on in the article, you will see some 2-digit numbers prefixed by a $ (e.g. $03). This is the ATARI standard method for defining a hexadecimal (base 16) number. The digits in the hexadecimal system are 0-9 and A-F (translating to 10-15 in decimal mode). By using hexadecimal numbers, it is possible to represent the numeric value of 1 byte in just 2 digits, making it easier to interpret it as individual bits (once you get used to the numbering system), plus it reduces the amount of space needed to display the number (2 locations instead of 3, since a byte can hold a decimal value from 0 to 255).

You will often find that 2-byte numbers and addresses on the ATARI are stored in low/high format (the value as we would understand it is 256*high+low). This is because that is the way the 6502 microprocessor chip (the heart of your ATARI) works with numbers and addresses. Many computers (including the human brain) work the opposite way (high/low), but not your trusty 6502.

And now, on with the show.

OVERVIEW

Standard disks (the term is DOS 2 disks) have 720 sectors, numbered from 1 to 720. These are laid out in 40 tracks (40 concentric rings on the diskette) of 18 sectors each. Enhanced density disks (1050 DUAL density) have 40 tracks of 26 sectors each, yielding 1040 sectors in total. Getting back to DOS 2 format disks, of the 720 sectors available, 13 are reserved for special functions:

1	initial boot sector
2	remainder of boot
3	garbage, but part of boot
360	VTOC
361-368	DIRECTORY
720	unused

This leaves 707 sectors free for use as data storage.

The first 3 sectors are always 128 bytes long, while the remainder are either 128 bytes for single density, or 256 bytes for double density.

SECTOR LAYOUT - DOS 2 FORMAT DISKS

DOS BOOT - SECTOR 1:

BYTE:	0	1	2 3	4 5	6 - 8	9
	xx	03	low \| high	low \| high		xx
	unused	no. sectrs to boot	boot load address	dos init address	jump to boot init code	max open files

BYTE:	10	11	12 13	14	15 16	17
Continued:	xx	xx	low \| high	xx	low \| high	xx
	active drives flag	unused	boot end address+1	DOS flag	starting sector of DOS.SYS	sector link offset

BYTE:	18 19	20 127
Continued:	low \| high	108 bytes of data
	DOS.SYS starting address	boot instructions

Some of the flag bytes deserve more explanation:

BYTE 10 - ACTIVE DRIVES FLAG - This is a set of bits that define which drive(s) are available for use by DOS.

```
BIT:   7 6 5 4 3 2 1 0
                    └── Drive 1
                  └──── Drive 2
                └────── Drive 3
              └──────── Drive 4
            └────────── Drive 5
          └──────────── Drive 6
        └────────────── Drive 7
      └──────────────── Drive 8
```

Typically, bit 0 will be on (giving $01), meaning drive 1 is available, or bits 0 and 1 will be on (giving $03), meaning drives 1 and 2 are available.

BYTE 14 - DOS FLAG - This byte flags whether DOS has been written on the disk (i.e. DOS.SYS exists on the disk), and if so, what density it is:
$00=no DOS, $01=single density DOS, $02=double density DOS

BYTE 17 - SECTOR LINK OFFSET - This contains the offset within a sector of the sector link information (described later). In single density, this will be $7D, and in double density will be $FD.

DOS BOOT - SECTOR 2:

BYTE:	0	1	2	3	4	5	6		127
	128 bytes of data (more instructions)								

DOS BOOT - SECTOR 3:

BYTE:	0	1	2	3	4	5	6		127
	128 bytes of garbage....ignore it								

VTOC - Sector 360:

BYTE:	0	1	2	3	4	5-9	10-99	100-127
	flag	low \| high		low \| high		unused	VTOC	unused
	dir. type	max sector no -garbage, ignore		no. free sectors			sectors used bits	

Table of bits to sector numbers:

bit	7	6	5	4	3	2	1	0
byte 10	-	1	2	3	4	5	6	7
byte 11	8	9	10	11	12	13	14	15
	.							
	.							
byte 99	712	713	714	715	716	717	718	719

A zero bit means that the sector is in use. Note that sector 0 has a slot reserved for it, and sector 720 does not. There was a communication mixup between the hardware developers and the authors of DOS. The hardware recognizes sectors 1 to 720 but the developers thought it was to be 0 to 719 (DOS was written without having a real machine to work on....I think Bill Wilkinson and friends did an admirable job, considering; BASIC REV. A was developed the same way).

Since sectors 1 to 3 are reserved, on an empty disk byte 10 should contain $0F. Then there will be a lot of $FF's until we get to byte 55 (which covers sectors 360 to 367) which will be $00, and byte 56 (which covers sectors 368 to 375) which will be $7F. The remaining bytes up to 99 will again be all $FF's.

The above structure makes calculating sector numbers/bit offsets fairly easy, since modulo 8 arithmetic can be used to convert back and forth, which is why Bill et. al. did it that way, and did not fix the problem later on, when it was uncovered. It would have been too difficult, I guess, so they left it alone.

On a double density disk, only the first 128 bytes are used. The remainder of the sector is ignored.

DIRECTORY - SECTORS 361 to 368

There are 8 sectors, with 8 entries in each (hence the limit of 64 files per disk). Each file entry is 16 bytes long, and is laid out as follows:

BYTE: 0 1 2 3 4 6 12 13 - 15

flag	low \| high	low \| high	name	extender
see below	sector count	starting sector		

FLAG VALUES: $80 = deleted
$40 = exists
$20 = locked
$01 = open for output

These bits can be merged together, allowing the following 5 combinations:

$80 = deleted
$60 = exists and is locked
$41 = exists and is open for output
$40 = exists
$00 = entry never used since disk formatted

Note that the directory only knows the first sector of a file. All other sectors must be found by reading the file. This explains why, when you delete a file, that you hear the disk merrily reading sectors for awhile. In order to add the sectors from that file back onto the free sector list (in sector 360), DOS must find all the sectors, and so has to spin through the whole file.

Since 8 entries at 16 bytes each yields 128 bytes in total, the last half of the sector is ignored on a double density disk.

USER DATA - SECTORS 4 TO 359 AND 369 TO 719

	BYTE: 0 ... 124	125 ... 126	127
Single Density	user data	file no./chain	count

	BYTE: 0 ... 252	253 ... 254	255
Double Density	user data	file no./chain	count

FILE NUMBER / CHAIN

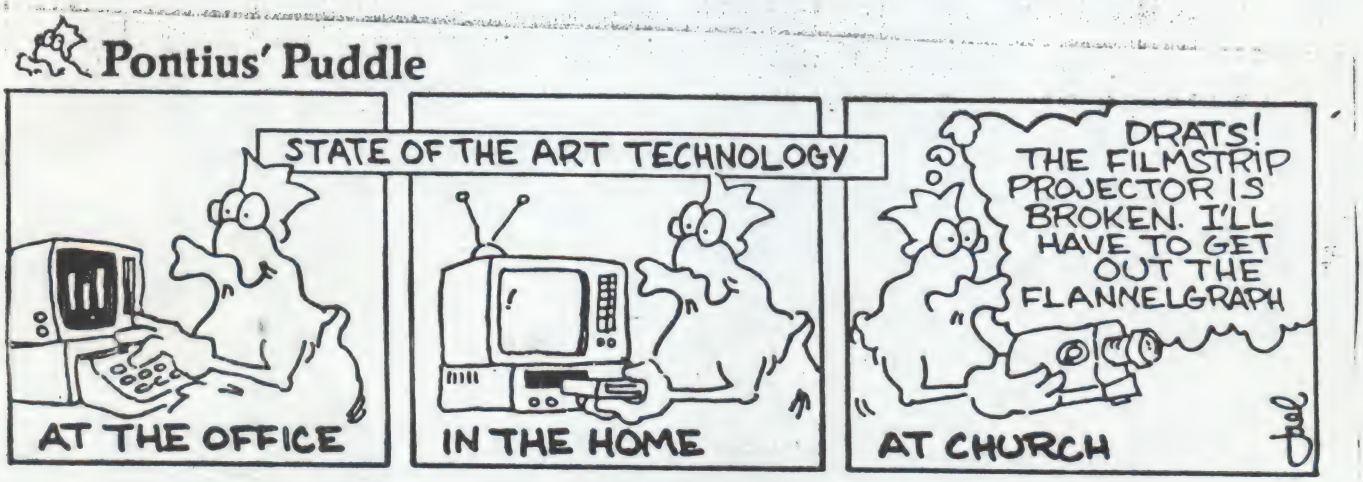
BYTE:	125 or 253.....								126 or 254.....							
BIT:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

6 bit file number (0-63)	10 bit next sector number (0-1023)

The file number relates to the relative position in the directory of the entry for the file owning this sector. If they don't match (i.e. the sector chaining has gotten messed up), you will get an ERROR 164 (file number mismatch) when processing the file.

The count field gives the number of bytes in use in the sector (1 to 125 in single density, 1-253 in double density).

By using a disk sector editor (several are available in the CALTARI library), you can examine your disks in more detail, and perhaps gain a better understanding of this information.



** PRODUCT REVIEW **

PERCOM DISK DRIVE with OS/A+ DOS
PART 1

By Ed Sisul

One day in mid-January, I arrived home from work to find 10 new Percom disk drives in my living room. After over a year of frustration with the less than spectacular 410 recorder (actually, four 410s), I conceded defeat and, along with nine other Atari owners, ordered a Percom disk drive. Three people purchased single-density drives, six puchased double-density drives, and one lucky soul purchased a dual-unit,dual-head,double-density drive. Mine was one of the double-density drives and I have been using it with my 400e (a 1 1/2 year old 400 enhanced with 48K, an add-on full stroke keyboard,and a GTIA chip, but no ROM B upgrade). To date, all of the single-unit drives have worked reliably - no problems whatsoever. One drive of the dual-unit failed after one month and it took about one month to repair the drive. The Percom appears to be fully compatible with existing software for the Atari. We have found no commercial software, heavily protected or not, that won't work on any of the Percom drives. In fact, even the problem of formatting disks with Letter Perfect discussed in the April '83 issue of ANTIC magazine doesn't seem to be a universal problem. I've had no problem formatting and using data disks with Letter Perfect on my system.

Why buy your drive from Percom rather than Atari ? Besides the obvious reason that Percom offers double-density drives and Atari doesn't, the Percom drives are designed to reduce the cost of expansion. The initial drive contains a controller that can drive up to three add-on drives. The add-on drives don't need a controller, consequently they cost $100 to $200 less than the master drive (but a $30 cable is needed to interconnect up to 3 add-on drives). Percom also offers dual units at less cost than the cost of one controller and one add-on unit. The dual-unit, dual-head, double-density drive doesn't cost much more than two 810s. In all, Percom offers six types of drives: a single-head/SD drive, a dual-unit, single-head/SD drive, a single-head/DD drive, a dual-unit, single-head/DD drive, a dual-head/DD drive, and a dual-unit,dual-head/DD drive. Each is available as a controller drive or as an add-on drive.

The Percom uses an upright drive unit (the disk rotates in the vertical plane rather than in the horizontal plane as it does in the 810). The drive unit is manufactured by Tandon. The Percom is heavier than the 810, mainly because it has a metal cabinet and a built-in power supply; but is considerably more compact than the 810. The Percom has two serial I/O ports in the back (13-pin MOLEX sockets just like the one on the side of the 400/800). One is used to connect the Percom to the computer. The other can be used to connect an 810 or another Percom controller drive to the Percom as a second drive. It can also be used to connect a 410 recorder to the computer through the Percom. There is a third connection in the back for attaching the add-on drive cable. The DD drives use a flat cable connector, but the SD drives extend the edge connector of a pc board through a slot in the cabinet as the connection point. The DD drives contain a 4-pole DIP switch in the back for drive number assignment (and for default density asignment) that are more convenient to use than the drive number switches on the 810, but the SD drive uses four bent pins (on the edge connector) that must be jumpered to configure the drive. The Percom uses a single LED indicator

the front panel that glows dimly (flickers) at turn-on, is off during standby, and glows brightly during read/write operations. The on/off rocker switch is located in the rear and is confusingly labled I for on and O for off on my drive.

So much for the mechanical aspects of the Percom drives. Now let's discuss the particulars of operation. Since the operation of the SD drive is virtually identical to that of the 810, I won't discuss the SD drive any further. The rest of this article will be devoted to discussion of the DD drive. The double-density drive is currently supplied with OSS OS/A+ DD DOS, Version 4.10. It can also be used with Atari DOS 2.0S, OSS OS/A+ SD DOS, and Atari DOS converted to double-density DOS (DOS 2.0P). To keep this review from taking up an entire newsletter, I'll review the special features of the double-density disk operating systems in detail in next month's newsletter. The double-density drive can read, write, or format in either the double-density mode or the single-density mode. When a formatted disk is booted, the drive automatically switches to the proper mode of operation. Unlike the 810 or the SD Percom, the DD drive uses an LED and a photocell to sense the index hole in the diskette when formatting a disk. Consequently, to format the backside of a disk, it is necessary to punch a set of holes in the disk jacket (in addition to punching a write-protect notch) for the flip side. A technique for doing this is described in detail in issue 10 of A.N.A.L.O.G. magazine. I've been using this technique since acquiring my drive (I always use both sides of a disk) and have had no problems. An alternative is to buy flippy-floppy disks, which are designed to be used on both sides in a single-head drive, but to me its not worth the extra cost. In the case of single-density disks, one other alternative exists. The index hole is only used to format the disk. It is not used for read/write operations. This means that if you have access to an 810 or a SD Percom you can format the backsides on one of these drives, then use them on the DD drive without having to punch the holes. I have had no short term or long term problems using both sides of single-sided disks for either SD or DD storage. Furthermore, I have had no problem formatting or using SD disks for DD storage (or, for that matter, using DD disks for SD storage). Just to be safe, however, I store all important DD files on DD disks.

Is the double-density capability worth the extra price of admission ? If you are a novice programmer and only use purchased programs, the answer probably is no; otherwise the answer is a definite yes. There are two main advantages to double-density: reduction of the number of disks needed to store programs and larger data storage capacity and faster data access for utility or business type applications. Obviously, since DD disks have twice the storage capacity of SD disks, only half as many disks are needed to store information in double-density. Unfortunately, not all programs can be easily moved to double-density. It's usually no problem to move BASIC files and binary load files to double-density, stacking as many to a disk as will fit. Many purchased programs, however, are machine language, boot load files and you will be stuck with one program to a disk unless you are a skilled hacker and can convert them to binary load files that will run in double-density. Even if you can't convert every program, the typical library will benefit from usge of double-density storage. For example, my current library consists of about 37 programs on 27 sides of SD disks and 152 programs on 9 sides of DD disks. You probably won't recoup the extra cost of the drive in disk

savings, but just being able to reduce the quantity of disks to have to store and sort through is worth the extra cost. There currently are no off-the-shelf application programs available that take advantage of the added data capacity and data access speed of the double-density system, consequently you will have to convert existing programs or write your own custom application programs to benefit from these capabilities. Hopefully, as more systems reach the field, the software houses will begin to generate programs specifically for double-density. ❀

LOAN AMORTIZATION TABLE GENERATOR

by Charles Robinson

Here's a program that no home computer should be without! No longer will you have to fret and worry over whether you can afford that new peripheral - now you can instantly know what your monthly payments will be!

I've checked the numbers versus my house and car loans and the numbers agree within a few cents so you can safely trust them (just don't go argue with your bank over the differences!).

The numbers go by pretty fast so either stop the program with a CTRL 1 or send it to your printer (if you have one) by changing the PRINT statement in lines 510,530,550, and 560 to LPRINTs.

(The original program was written in DEC's FOCAL mini-computer language.)

```
1 REM ********************************
      *  LOAN AMORTIZATION TABLE    *
      *                             *
2 REM *   ORIGINAL IN DEC FOCAL     *
      *       Translated By         *
      *     Charles Robinson        *
3 REM *   JUNE '83 AURA OUTPUTS     *
      ********************************
10 ? CHR$(125):? "        LOAN AMORTIZA
TION TABLE":? :? :? :?
20 ? "ENTER INTEREST IN PERCENT";:INPU
T J
30 ? "ENTER AMOUNT OF LOAN";:INPUT A
40 ? "ENTER NUMBER OF YEARS";:INPUT N
50 ? "ENTER # OF PAYMENTS/YEAR";:INPUT
 M
100 J=J/100
110 N=N*M
120 I=J/M
130 B=1+I
140 R=A*I/(1-1/B^N):R=INT(100*R+0.5)/1
00
200 ? :? :? "MONTHLY PAYMENT IS ";R
210 ?
220 ? "TOTAL INTEREST IS ";R*N-A
230 ? :?
240 ? "PRINT TABLE? (0=NO, 1=YES) ";:I
NPUT T
250 IF T=0 THEN 1000
500 B=A
510 ? "↖":? :? "INTEREST  APP TO PRIN
  BALANCE":? :?
515 POKE 201,12
520 L=B*I:P=R-L:B=B-P
525 L=INT(100*L+0.5)/100:P=INT(100*P+0
.5)/100:B=INT(100*B+0.5)/100
530 ? L,P,B:?
540 IF B-R>0 THEN 520
545 FI=B*I:FP=R-B*I:FI=INT(100*FI+0.5)
/100:FP=INT(100*FP+0.5)/100
550 ? FI,FP:? :?
555 LP=B*I+B:LP=INT(100*LP+0.5)/100
560 ? "LAST PAYMENT! ";LP
570 POKE 201,10:? :? :?
580 ? "ENTER 0 FOR NEXT PROBLEM,
       ENTER 1 TO END";:INPUT Z
590 IF Z<1 THEN 10
1000 END
```

PRODUCT REVIEW

PERCOM DISK DRIVE/OS/A+ DOS - PART 2

by Ed Sisul

In the JUNE newsletter, I reviewed the new disk drives from Percom. As promised, this month I will review the double-density operating systems available for the Percom.

The DD drive is currently supplied with OS/A+ version 4 from Optimized Systems Software (OSS). My drive came with version 4.0D, which has since been superseded by version 4.1D. Two other DD operating systems are available for use on the Percom: OS/A+ version 2 and Atari DOS 2 converted to double-density.

Before OS/A+ version 4 was available, the Percom DD drive was supplied with a disk containing two utilities: one, called BLD, to convert DOS 2.0S to double-density DOS 2.1P; and one called SDCOPY, to transfer files between DOS 2.0S and DOS 2.1P. This disk is still available and can be purchased from Percom for about $30.00. It's worthwhile to acquire this utility if you want to use any of the "canned" disk utilities (such as AUTORUN.SYS files) found in magazines with double-density since most of them, as written, will work with DOS 2.1P but not OS/A+.

OS/A+ was developed at OSS by the authors of Atari BASIC and Atari DOS as an enhanced disk operating system. It originated as version 1 for single-density applications. Version 2 was developed to access up to 256K bytes per disk and supports single or double density. Version 4 was developed to access up to 15M bytes per disk with up to 512 bytes per sector. On the Atari, version 4 can only be used with the double-density drive. Version 1 and version 2 provide the maximum file compatibility with DOS 2.0S (all three use the same FMS). Version 4, however, uses a mapped file structure (instead of the linked file structure used by DOS 2.0S and versions 1 and 2) to provide fast true random access files (you can POINT without the need to have previously NOTEd). The overview of OS/A+ that follows is based primarily on version 4. However, most of the features of version 4 are available on version 1, so those of you interested in an enhanced single-density operating system can benefit from the discussion.

OS/A+ uses a command driven system that provides features and conveniences not available with Atari menu DOS. OS/A+ uses three classes of commands called intrinsic commands, extrinsic commands, and execute commands. The intrinsic commands access memory-resident, commonly used disk utilities; the extrinsic commands load disk resident utilities (mostly copy programs); and the execute commands are a subset of commands that provide for batch execution of OS/A+ commands from a file.

The intrinsic commands are loaded into RAM as part of DOS.SYS when the disk is booted (no DUP.SYS in OS/A+). Typing DOS accesses these utilities without disturbing any user program in memory (no need for MEM.SAV with OS/A+). When DOS is entered, the screen is cleared and the prompt "D1:" appears as a reminder of the default drive and to accept a command. There are 13 intrinsic commands which may be given by typing in the first three letters of the command name. They are:

CARtridge - Re-enter the cartridge.
DIRectory - List the disk directory to the screen or the printer.
END - Stop execution (for batch processing).
ERAse - Delete a file.
LOAd - Load a binary file into user memory.

Screen - Turn off command echo (for batch processing).
PROtect - Lock a file.
REMark - Print remarks on the screen (for batch processing).
RENname - Change the name of a file.
RUN - Transfer control to an address in memory.
SAVe - Save a portion of memory to disk.
SCReen - Cause batch commands to be echoed to the screen (for batch processing).
TYPe - Print ATASCII text files to the screen, printer, or another disk file (version 4.1D only).
UNProtect - Unlock a file.

All of the commands (other than the special batch processing commands) function similar to their DOS 2.0S counterparts. Version 4 allows filenames of up to 30 characters in length. The intrinsic commands and the double-density capacity consume about 4K of free RAM. Consequently, OSS recommends using OS/A+ on a system with at least 32K of RAM. The RAM/disk space consumption of DD verses systems is shown below.

TABLE 1

OPERATING SYSTEM	DOS FILES	DISK SECTORS	FRE(0) 48 K
DOS 2.0S	DOS.SYS	39	32274
	DUP.SYS	42	
	OVERHEAD	13	
OS/A+ V.1	DOS.SYS	43	31758
	OVERHEAD	13	
DOS 2.1P	DOS.SYS	19	28818
	DUP.SYS	21	
	OVERHEAD	13	
OS/A+ V.4	DOS.SYS	30	28430
	OVERHEAD	21-23	

The extrinsic commands load and run special disk resident utilities that have the file extension .COM (e.g. COPY.COM). There are nine fundamental commands supplied with versions 2 and 4. They are executed by typing in all of the letters preceding the .COM extension. They are:

ADOS - Installs version 2 along with version 4. This allows access of DOS 2.0S and version 4 disks while transferring files. Used in conjunction with COPY24. (13 sectors)

CONFIG - Used to report the status of all drives and to change the status of configurable drives. (19 sectors)

COPY - Used to copy files (except between version 2 disks and version 4 disks). Includes options to force overwrite of an existing file, query before each file transfer, display prompts when copying on a single drive, and wait for proper disk insertion. (37 sectors)

COPY24 - Used to transfer files between version 2 disks and version 4 disks. This is how files are transferred from single-density to double-density and vice-versa. (21 sectors)

DO - Used to perform several operations on one command line. (2 sectors)

DUPDBL - Used to copy entire version 2 double-density disks (version 2 only).

DUPDSK - Used to copy entire version 2 SD disks or version 4 DD disks. (6 sectors)

HELP - Provides a menu of system commands to help the beginning user. (15 sectors)

INIT - Used to initialize (format) a disk. (12 sectors)

RS232 - Installs the serial device handlers ("Rn:") for use with the 850 Interface Module. (1 sector)

The number in parenthesis after the

explanation is the number of disk sectors occupied by the program on a version 4 disk. These commands of course can only be invoked when a disk (usually the master system disk) containing the particular program is inserted in the drive and, since the program is loaded into RAM, any user program present is erased. Any DOS-compatible load file can be made into an extrinsic "command" by affixing the .COM extender to its filename.

The execute feature allows execution of one or more commands with a single command. Any text file with the filename extension .EXC can be used as an execute file.

Each line of the file must be a valid OS/A+ command. Batch processing is initiated by typing in the filename, less the .EXC extension, followed by the symbol "@". If the file is named STARTUP.EXC, it is automatically run upon boot of the disk (equivalent to AUTORUN.SYS of DOS 2.0S).

OS/A+ also supports all BASIC disk commands such as RUN, SAVE, LIST, LOAD, ENTER, PUT, GET, etc. The LOAD or RUN commands execute from OS/A+ version 4 at about the same speed as from DOS 2.0S, but SAVEs and ERAses (DELETES) are executed much faster in version 4. Table 2 below compares the execution times of a 26K (of code) BASIC program (GAMBLER from SOFTSIDE MAGAZINE).

TABLE 2

	OS/A+ V.4	DOS 2.0S
DISK SECTORS:	100	207
LOAD TIME(sec.):	23	22
SAVE TIME(sec.):	23	66
DELETE TIME(sec.):	4	24

Copying of files or entire diskettes in double-density works essentially the same as it does in single-density. Copying a full Version 4 double-density disk (DUPDSK) requires six passes (12 disk swaps with one drive) and takes about seven minutes. Transferring the contents of a SD disk to a DD disk or from a DD disk to SD disks is a more difficult task. It must be done one file at a time, using COPY24 with OS/A+ or SDCOPY with DOS 2.1P. This is especially slow and tedious with OS/A+ if the disks contain lots of files, because it almost always takes two passes (four disk swaps) for each file to be transferred.

OSS boldly states in the OS/A+ manual that "we firmly believe that the command system of the OS/A+ Console Processor (CP) is superior to a menu approach". How can the command system be simpler or easier than typing a single letter from a menu (ala DOS 2.0S), you might ask? Is OS/A+ really that much better? After learning to use both systems simultaneously, I certainly feel that OS/A+ is a better way to go. To quantify this, I've put together a scenerio based on my experiences, which (although a bit extreme) serves to illustrate the advantages of the command approach.

After an hour of typing in another installment of the great epic program, GHOSTSTOMPER, from FRANTIC magazine, it's time to stop and save it to disk. I can't remember which version I've just completed and I'm not sure how much room is left on the disk, so I'll have to access DOS. Using Atari DOS the entire save procedure would go something like this:

1. Type DOS [RET] - Wait... wait... wait... for MEM.SAV to do its thing (used this time to thank God that MEM.SAV was on this disk).

2. Type A [RET][RET] for directory listing - See "GHSTSMP.V2" as last entry followed by "5 FREE SECTORS." OK, I'll have to erase GHSTSMP.V1 to make room for V.3.

Let's see, do I type E for erase or ?? - DRAT! The menu was scrolled off screen by the directory listing.

3. Type [RET] for menu - Ah-Ah! D for delete.

4. Type D [RET] - See "DELETE FILE SPEC". Oh no! I forgot what abbreviation I'd concocted for GHOSTSTOMPER and the menu has erased the directory from the screen.

5. Type A [RET] - Wait... wait... Oops, [RET] again - See "GHSTSTMP.V1".

6. Type D [RET] (I remembered it now)

7. Type GHSTSTMP.V1 [RET] (Good thing I wrote it down at step 6 because I can't see the bottom of the directory listing anymore).

8. Type Y [RET] - I'll assume its gone.

9. Type B [RET] for cartridge (This one isn't hard to remember) - Wait... wait... wait... for MEM.SAV to disgorge.

10. Type SAVE"D:GHSTSTMP.V3 [RET] - I guess I'd better lock the file so I don't blitz it later. - Here we go again.

11. Type DOS [RET] - Wait... wait... wait...

12. Type A [RET] [RET] to make sure V.3 is really there.

13. Type L [RET] for lock - What do you mean "LOAD FROM WHAT FILE?"! - Oh!

14. Type [BREAK] [RET] - See "F. LOCK FILE"

15. Type F [RET]

16. Type GHSTSTMP.V3 [RET]

17. Type B [RET] - Wait... wait... wait...

18. Catch breath. Stretch Fingers. Proceed.

Using OS/A+ version 4, the whole procedure almost always would go like this:

1. Type DOS [RET]

2. Type DIR [RET] for directory listing - See GHOSTSTOMPER.V2 as last entry followed by 5 FREE SECTORS.

3. Type ERA GHOSTSTOMPER.V1 [RET] (The directory was still visible on the screen).

4. Type CAR [RET] to return to BASIC - Program still in memory.

5. Type SAVE "D:GHOSTSTOMPER.V3 [RET].

6. Type DOS [RET]

7. Type DIR [RET]

8. Type PRO GHOSTSTOMPER.V3 [RET]

9. Type CARP [RET] (Just having some fun. Only the first three letters count).

10. Proceed.

As you can see, OS/A+ eliminates four drawbacks to Atari DOS: MEM.SAV, the eight character limit for filenames, the need to recall abstract one-letter codes, and the idiosyncracies of the menu/directory displays.

So, in closing, I find that the Percom disk drive in conjunction with OS/A+ is not only more powerful than the Atari disk system, but also is a better value for the money.

AtariWriter: Product Update

By Atari, Inc.

AtariWriter is the new word processing software now available from Atari. There are a number of differences between the old Atari word processing program and the new AtariWriter.

The new AtariWriter software is considered FILE oriented, while the old word processing program is considered page oriented. The formatting and paginating time with a lengthly document is greatly reduced over the old word processing program, because AtariWriter formats a document as it prints.

AtariWriter is available in cartridge form, not diskette, so document files can be saved to cassette as well as to diskette. This means more memory is available to write and edit with. Also, no more worries about the Master disk going bad! It may be used with either the Atari 400, 800, 1200XL, 600XL, or 800XL computers.

Even though AtariWriter has a 40 column screen output, it compensates for this limitation with a unique print preview feature. Print preview allows the user to view the 80 column format on the screen before the document is printed. The use of the ESCAPE key anytime during the writing/editing of a document will take the user back to the menu with no harm to the file or system. Also, the SYSTEM RESET key can be used to return the user to the program menu with no harm to the file or system.

AtariWriter is extremely user-friendly and is compatible with any non-ATARI parallel printer. The user can easily insert the control codes for their printer anywhere into the text, thus giving them all the special functions of their non-Atari printer. This puts an end to printer/program incompatibility. Also, the AtariWriter can be merged with document files from the old Atari word processing program.

Control Codes. The AtariWriter Word Processor may be used with any parallel (or "Atari serial") printer. The AtariWriter program includes printer drivers for four Atari (820 and 822 40 column and 825 and 1025 80 column) printers.

If you intend to use AtariWriter with any printer other than those listed above, you will need to obtain a printer driver or use your printer's control codes to format and print your documents. The following steps illustrate how to enter printer control codes into your AtariWriter text.

1) Select the proper decimal control code sequence from your printer manual for the function you want to perform. (You may have to covert the hexidecimal values to decimal.)

2) Enter the control code sequence at the desired place in your AtariWriter document. The codes are entered by typing "control 0" before each decimal number in the cotrol code sequence. Control codes that affect the entire document should be placed at the top of the text immediately under the AtariWriter's format codes. Control codes that affect only portions of the text such as underline and double width (elongated) print should be inserted in the text at the point where the function begins.

3) When you use the "Print" or "Print Preview" function the first time after you load AtariWriter, the names of four Atari printer drivers will appear at the bottom of the screen. If you do not have one of the printers listed, select the 3rd printer (Atari 820) driver and then print or preview your document.

Example. If you wanted to use elongated print and your printer's decimal control codes were 14/117 to start elongated printing and 14/118 to end elongated printing, you would do the following to print an elongated heading called EXAMPLE:

1) Hold down the key marked "CTRL" with one hand and with the other press the "0" key. This will put the inverse video letter "0" on the screen. Then type the first decimal number "14". Again hold down the "CTRL" key while pressing the "0" key and then type the number "117". Now type the word "EXAMPLE" and then repeat the process using the control key, control 0, and the numbers 14 and 118. Your screen should now look like this (underlined characters on the paper represent inverse video characters on the screen):

0140117EXAMPLE0140118

2) Note: A "CONTROL O" must be placed in front of EACH decimal number. The "RETURN" key msut be pressed somewhere in the document after the point in which the codes were inserted.

3) On those printers that have international characters you must first enable the printer's character set (using "CONTROL O / decimal code" sequence outlined above) and then repeating the sequence for each character to be printed.

4) The control codes will not print on your printer and will not take up space unless directly followed by pressing "RETURN" in which case they will appear as a blank line on your printer.

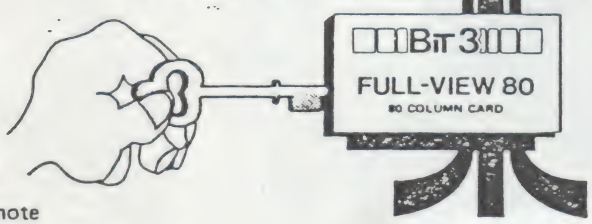
Using the method above, you should be able to obtain almost any capability that your printer can perform. Do not try to mix type styles on the same line (such as elongated and 10 cpi). AtariWriter has no way of knowing what codes you are sending to your printer and therefore cannot make the spacing adjustments to compensate for the change in print styles. This should not be a problem if you have a printer driver.

Printer Drivers. The purpose of a printer driver is to convert the printing and formating codes in AtariWriter to codes that your printer understands. Using a printer driver with AtariWriter will allow you to use the printer codes in the AtariWriter manual. For all printer functions supported by AtariWriter you will not need to use the "CONTROL O / decimal code" sequence mentioned above. Printer drivers are now available for the following third-party parallel printers:

Atari 1020 and Atari 1027
Epson MX-80, FX-80, and MX-10
Epson MX-80, MX-100 w/GRAFTRAX
EPSON MX-80, MX-100 FT III
GEMINI-10
NEC-8023A
C. Itoh PROWRITER-8510
IDS 480
Mannesman Tally
Okidata Microline 92A and 93A

Disks retail for $24.95 and each disk contains a file for every printer listed above.

YOUR KEY TO ATARI® 800 ENJOYMENT



Get professional 80 column word processing, data management, remote terminal configurations, plus many more applications thanks to the BIT 3 FULL-VIEW 80. Clear crisp characters on your CRT monitor with full descenders are standard. And all your current Atari 40 column and graphics programs work: switch between normal Atari 40 column/graphics mode and 80 column mode either under program or keyboard control. Just plug into slot 3, cable up and fun. Maintain 48K RAM capacity by installing the BIT 3 32K MEMORY PLUS RAM card in RAM slot 2.

LJK's Letter Perfect gives you an 80-column word processor using our Full-View 80. Other popular software that runs with the FULL-VIEW 80 is Atari's Assembler Editor, Basic Cartridge, Macro Assembler, and PASCAL; Optimized System's Basic A+, OS/A+ and EASMD; Microsoft's BASIC; Teletari by Don't Ask Computer and T.H.E. Smart Terminal by Binary Computer Software. More applications are being added every day including CP/M by SPW.

See a review of the FULL-VIEW 80 in A.N.A.L.O.G. Computing, Issue 9, page 31. Other reviews available upon request.

Full-View 80 $250.00
32K Memory Plus $64.00

OTHER FINE PRODUCTS FROM BIT 3

For Apple computers: Full-View 80, 2 Serial Channel Card. For IBM PC computers: Multibus Expansion; General products: RS232 Gender Changer/Pin Reconfigurator.

CONTACT BIT 3 OR YOUR DEALER FOR PURCHASE.

BIT 3 COMPUTER CORPORATION
8120 Penn Avenue South
Minneapolis, Minnesota 55431
612—881-6955

VISA-M/C

IMPORTANT FOR RICHARD 5-8-85

A USE FOR PERCOMS FROM DR. DOWNLOAD

BBS 614-587-3774

Many callers to the Dr.Download BBS are amazed by the sheer volume of d/l files. "Surely you must be running an ATR8000 with 8" drives!" they say. Not so! A more elegant way to construct a large data base is to acquire a Percom controller card. These magical PCBs enable Atari users to connect any 34-pin Shugart-compatible drive mechanism, such as Tandon 100-2, Teac, MPI, etc. to their systems. The term "Percom drive" is a misnomer: Percom did not make drives! They made controller cards, boxed them with power supplies and MPI drive mechanisms, & sold $150 of hardware for $450 (for a SSDD model RFD40) or for $750 (for a DSDD model RFD44). The controller cards in these models were identical. The difference was that the model 44 drive mechanism had two heads, a feature that must have cost Percom an additional $15. The slim volume of documentation that accompanied these models mentioned that "Percom slave drives" could be connected to the 34-pin plug, and seemed to imply that these "slaves" were specialized hardware. The fact is that any generic disk drive can be connected to an Atari computer using a Percom controller card. Percom cards with model numbers RFD44, RFD40, and AT88-SPD feature the 1795 disk controller chip that is designed for SD, DD and DSDD operation. The cards have two standard Atari serial ports, and a 34 pin parallel plug (RFD40, -44) or edge-card connector (AT88). Naturally, our interest is in the 34-pin connector, through which any any four Shugart-compatible drives can be wired in parallel. There is nothing special about the drive mechanism that comes in the box: it is connected in parallel also, and may be removed or replaced. So, lets say you have an AT88 that came with a SSDD half-height MPI mechanism, and that you want to replace this mechanism with four Tandon 100-2 DSDD drives (giving you 1.5 Mb of storage for about $600-not a bad deal). Pull out the MPI unit & sell it to someone you don't like (that particular model gives problems). Use standard plugs, connectors & ribbon cable to wire the Tandons to the edge card connector on the AT88 controller card. The analog board of each drive features a jumper socket for selecting the drive number of that unit. Jumper each drive as 0, 1, 2, and 3, respectively. The physical location the drives with respect to the card is unimportant, as is the type of drive. SD and DD drives may be mixed & matched as desired. The new Percom ROM from Wordmark Systems, Garland, Texas, allows use of 8" drives as well! A true fanatic could thus connect four 8" DSDD units to make a 6 Mb data base! Dr.Download uses four Mitsubishi 4853 drives. These units are 5", half-height, 96-TPI cuties that can write 80 tracks per side, storing 720K per Atari-formatted disk. At $150 per drive, plus $40 each for two power supplies, plus $60 each for two dual enclosures, a 3Mb data base was created for less than $700. All of this hardware was advertised in Computer Shopper The DOS you use will make you or break you. Dr.Download uses MYDOS 3.013, a menu-driven DOS that looks comfortably like dear old Atari 2.0S, but is actually a much more powerful system that will allow you to run any combination of SS, DS, SD, DD, 5", 8", 40 track or 80 track units completely transparently. We use MYDOS to software-configure each drive for SD or DD, SS or DS, 40 or 80 track operation. Prospective users should be aware that the nitty-gritty details of exactly how things work with each type of drive depend on the ROM in the controller card. The Percom ROM went through many revisions, and no one version seems to do everything correctly for every type of drive mechanism. For example: the RFD40 card that runs Dr.Download uses the 1.1 ROM, which reads and writes to 80-track drives, but will not format them! The AT88 card, which uses the 1.2

PERCOM cont.

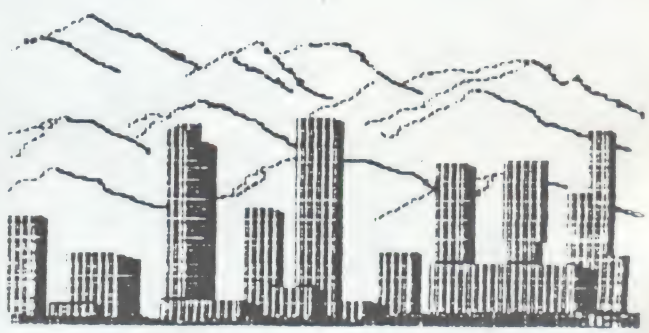
ROM, will read, write & format on 80 track drives, but has the annoying habit of forgetting the SD/DD settings for the upper 3 drives on booting. Needless to say, Dr.Download has an AT88 controller card operating offline to format disks for the BBS, and has 100 formatted disks in storage against the time that the AT88 card dies. Which brings up another important topic: the reliability of Percom service & support, and the longevity the controller cards. Percom no longer sells drive systems for the Atari. They do service the old drive systems, and if you are planning to build a BBS around an obsolete controller card you will want to know about your last line of defense. The Percom drive system is an example of a teriffic concept crippled by miserable engineering. Percom crammed the whole works into an enclosure that was too small and inadequately ventilated. The result often was that the card slowly fried over periods of extended use. The controller card for a BBS must therefore be enclosed separately from the drives it services to keep it cool & comfortable. If it dies anyway, send it to the Percom hospital in Dallas for replacement, and keep replacing dud cards until you get one that works. When the card works in a cool environment it will run indefinitely. The Dr.Download BBS controller card has been operating continuously since August 1984 without missing a beat. The Dr.Download backup controller card has been replaced four times in the same interval, so you just can't tell by looking. The Percom hospital in Dallas is a strange place from which drives may be returned in a week, or in 10 weeks, depending on nothing that you can see. If you are in dire need of your card or drive, phone it in as a rush order and it will return within 5 days for a 10% surcharge (an extra $5 on the $55 charge for replacing a controller card, for example). They will not sell you an extra card; they profess to know nothing whatever about anything discussed in this essay, and are likely to lie to you if you ask the innocent question, "Can I use any drive as a slave for a Percom, or do I need a Percom slave?". So thats the story about Dr.Download's 3 Mb data base. Callers will find it crammed with 26 menus of the best public domain music, art, graphics demos, pinball games, fancy fonts, Jean Rowe BASIC music, programs for the 1020 plotter, telecomm programs, and complete listings from ANTIC, ANALOG and COMPUTE! magazines. Eric Anderson, managing editor of the Piqua, Ohio DAILY CALL, writes essays on Atari topics in the ANDERNET column, and furnishes the BBS with Product Reviews that pull no punches. Meanwhile, call Dr.Download 24 hours daily at 614-587-3774, 300 or 1200 baud. No password, no time limit..... just huge phone bills.

*MILE HIGH ATARI
*7685 S DATURA CIR
*LITTLETON, COLO
* 80120

BULK RATE
U S POSTAGE
PAID
PERMIT#3417
DENVER CO

*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*

West LA Atari Users Group
Richard L. Reaser
5611 Sherbourne Dr.
Los Angeles, CA
NY.FMA 90056



OSS & ICD join forces

DATELINE: 1-27-88

Good news for Atari owners! ICD, Inc. and OSS, Inc. (Optimized Systems Software) have just signed an agreement which will "add new life to all Atari computers". Under this agreement, ICD will add the manufacture, marketing, and support of all current OSS software titles for Atari computers to ICD's already powerful Atari product line.

ICD's friendly, helpful staff is now trained in all aspects of support for the complete OSS product line including:

MAC/65
MAC/65 Toolkit
ACTION!
ACTION! Toolkit
BASIC XL
BASIC XL Toolkit
BASIC XE
Writer's Tool
DOS XL
Personal PASCAL for the Atari ST

Ordering and support for ICD/OSS products are available by calling 815/968-2228 from 8AM to 5PM CST Monday through Friday. Support is also available 24 hours a day 7 days a week on the following Electronic Bulletin Boards: ICD/OSS BBS 815/968-2229 300-9600 baud, CompuServe, GEnie, Delphi, and BIX. ICD's 24 hour FAX is connected to 815/968-6888.

The San Jose, CA numbers for OSS will no longer be providing support.

Orders for OSS products may be placed now. All products will be shipping in quantities soon. For more information call any of our product support lines or write to:

PAGE wned 2-88

ICD/OSS
1220 Rock Street
Rockford, IL 61101

Reading the ST Mouse on an XE

By Jeff Cleveland

Those people who own both an ST and an 8-bit probably already know that the ST mouse has the same connector as Atari joysticks. So, I plugged my mouse into my 130XE and wrote a program to try to read it.

I first tried reading it from the paddles 0 and 1, which are both on port 1, but got no reading. No matter how much I moved the mouse, the paddle values didn't change. That meant that the ST mouse is a digital mouse, not analog.

I then modified the program to print values returned from the STICK function in Action! (or Basic). The mouse didn't return regular joystick values for the eight directions. Instead, it returns a 2-bit value for each axis, X and Y. The X axis is in the first two bit of the returned value and the Y axis is on the third and fourth bits as follows:

Bit number: 87654321
STICK result: 00001111
bits 1 & 2: X axis
bits 3 & 4: Y axis

The direction the mouse has been moved can be found by

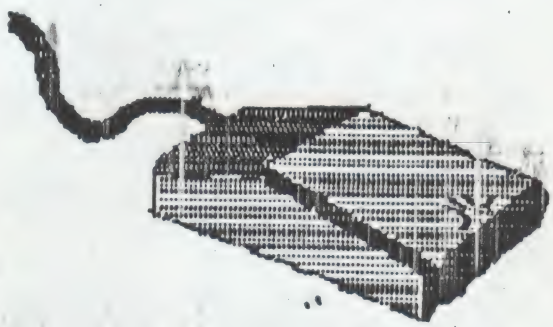
comparing the current reading with the past reading. Each time the mouse moves, only one of the bits changes value. Values proceed in this order: 00 01 11 10. By knowing what the last value was, direction can then be obtained by finding out if the new value is above or below the old value.

For example: If the stick function returns a 1 for the Y axis, and the next reading returns a 3, then the mouse has been moved up, because 3 is above 1 in the list (in a positive direction), and up is a positive direction for the Y axis. Notice that the values are not in numerical order. This eliminates the use of an if--then less-than greater-than comparison. The following Action! procedure uses an array of numbers to compare to in order to find direction.

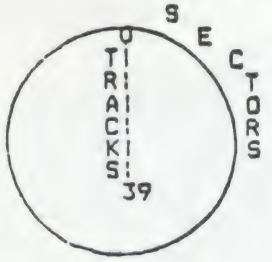
```
PROC mouse()
BYTE ARRAY mser=[ 2 3 1 0 ]
BYTE i,dx1,dx2,dy1,dy2

dx1=STICK(0)&3                ;and
with 3 to wipe out Y bits
dy1=(STICK(0)&12)  RSH  2 ;wipe
out  X bits and shift Y bits to
X position
dx2=STICK(0)&3              ;second
reading
dy2=(STICK(0)&12) RSH 2
IF dx1<>dx2 THEN
  i=0
  WHILE   dx1<>mser(i)    DO
;find first reading
    I==+1
  OD
  IF i<>0 THEN
    IF mser(i-1)=dx2 THEN
      PRINT("left")           ;put
you  left  code  or  proc  call
here
    ELSE
      PRINT("right")          ;go
right
    FI
  ELSE
;wraparound end of array
    IF mser(3)=dx2 THEN
      PRINT("left")           ;go
left again
    ELSE
      PRINT("right")          ;go
right again
    FI
  FI
FI
IF dy1<>dy2 THEN              ;now
check for up-down
  i=0
  WHILE   dy1<>mser(i)    DO
;find first reading
    I==+1
  OD
  IF i<>0 THEN
    IF mser(i-1)=dy2 THEN
      PRINT("up")             ;go
up
    ELSE
      PRINT("down")           ;go
down
    FI
  ELSE
;wraparound end of array
    IF mser(3)=dy2 THEN
      PRINT("up")             ;go
up
    ELSE
      PRINT("down")           ;go
down
    FI
  FI
FI

RETURN
```



DISK SECTOR STRUCTURE



40 Tracks (0-39)
18 Sectors/Track = 720 (0-719)
Track 0 = Sectors 0-17 or 1-18
Track 39 = Sectors 702-719 or 703-720
DOS can only handle Sector Numbers from 0 to 719
Disk Drives will only accept commands for Sectors 1-720

Disk Directory	8 sectors	361-360
VTOC - Volume Table of Contents	1 sector	360
FMS - File Management System	3 sectors	1-3
	707 sectors	128 Characters/Sector
Total	719 sectors	

DIRECTORY SECTOR - Names and Information for all files on disk-8 Sectors (361-368)

Each Entry - 16 Bytes. 8 entries on 1 sector (16 * 8 = 128 bytes)
8 sectors * 8 = 64 entries total

For each Entry:

Byte

0	1	2	3	4	5 - 12	13 - 15
Status Byte	Length (in sectors) LSB	Length (in sectors) MSB	Start Sector LSB	Start Sector MSB	Filename (primary) or space	Ext or Space

Bits

7	6	5	4 - 2	1	0
File Deleted	File In Use	File Locked	Spares	DOS-2 File	File Open for Output

VTOC - Keeps track of which sectors are being used - Sector 360

Byte

0	1	2	3	4	5 - 9	10 - 99	100 - 127
Use Byte	Total # of Sectors LSB	Total # of Sectors MSB	Total Free Sectors LSB	Total Free Sectors MSB	Not Used	Sector Use Map	Not Used

Sector Use Map - Byte 10 to Byte 99:
Each Bit of each Byte represents one sector on the Disk
If the bit is 0, then that sector is being used
If the bit is 1, then that sector is free
Bit 7 of byte 10 represents Sector 1, etc, to Bit 0 of Byte 99
which represents Sector 720

DATA SECTORS - Saved or Listed Programs, Text files, Data Files, etc, - 128 Bytes

Byte 0 - 124 - Actual Data
125 - 127 - Sector Identity Data (Link Data)

Byte / Bits

125								126								127			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				
								512	256	128	64	32	16	8	4	2	1		
File # This Sector belongs to								Next Sector of this File								# of actual bytes in This Sector (0-124)			

BINARY FILE HEADER - First 6 bytes of First Sector

Binary File Data - Created with Binary Save Option:

Byte

0	1	2	3	4	5
FF Binary File Definition Bytes	FF	Start Address LSB	Start Address MSB	End Address LSB	End Address MSB

Boot Sectors - No Cartridge or DOS:

Byte

0	1	2	3	4	5
0 by Tradition	# of Sectors to load	Load Address LSB	Load Address MSB	Address to Start Executing LSB	Address to Start Executing MSB

RIACE 2-87

Kyan Strikes Back!

by A Reasonably Informed Source

Chapter One

WHAT HAS GONE BEFORE: (version 1.1)

A long, long time ago, in a galaxy far, far away, KYAN unleashed an Atari (8-bit) version of KYAN PASCAL upon an unsuspecting universe, which soon became the most powerful Atari 8-bit PASCAL programming language in the cosmos.

This PASCAL did have two weaknesses, however; which allowed the rebel programmers to attack it.

This is the story of these weaknesses, and the steps that the KYAN empire took to counter them...

* VERSION 1.1 *

The two major problems with KYAN PASCAL V1.1 were: the poor manual and the lack of DOS accessibility, both within the programming environment and from the program itself.

These two problems made a fine PASCAL programming environment very difficult to work with. Let's take a look at what the problems actually were.

* THE MANUAL *

As I have stated many times before, the KYAN PASCAL manual was the WORST manual that I have ever worked with. It was basically the manual for the Apple computer version with brief notes pertaining to the Atari version. These notes were often unclear and sometimes inconsistent. Typos in strategically located positions, downright incorrect information, and lack of actual implementation examples made it difficult to learn how to edit and compile, not to mention learn new programming methods.

The resulting trial and error programming in a compiler-based language does not make for a very steep learning curve.

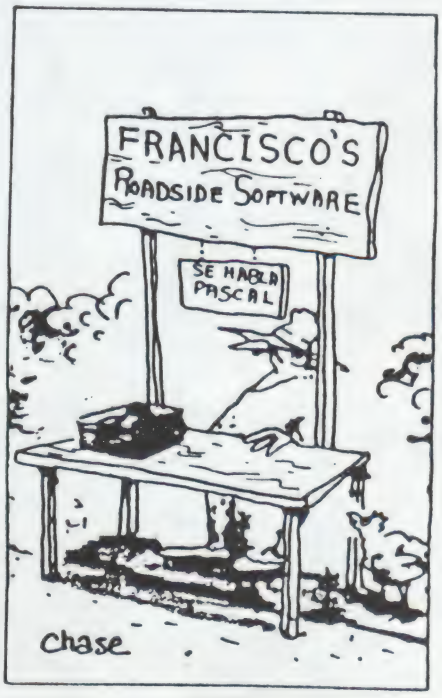
When programming examples were given, they were usually either too simple or too complex to be of practical use.

At one point, the Atari instructions were so bad, I actually had to refer to the Apple section and play it by ear!

All in all, the manual made the learning and implementation of PASCAL a real chore. If it were not for the excellent SAM'S PASCAL PRIMER (see RIACE REPORTER #26, "Kyan Pascal am's Pascal Primer"), I don't think I would have gotten very far with PASCAL at all.

The KYAN PASCAL manual was approximately 108 pages; PASCAL is a tough language to fit into 108 pages. Also included were a few Atari-specific addendum pages. This was all bound in a clear plastic binder of acceptable quality.

* DOS ACCESSIBILITY ILE MANIPULATION *





(continued from page 7 Kyan)

PASCAL was two-fold. First, the non-accessibility of the DOS commands from within the programming environment. To format a disk, lock or unlock a file, move a file to the ramdisk, or read a disk directory, you had to go to DOS, do what you had to do, and then BINARY LOAD (L) the PASCAL environment again. As often as I access the directories, this was a pain in the you-know-what.

The other difficulty arose from the fact that there was no way to access directories from within an executing program. As my programs tend to do a lot of text file manipulation, combined with PASCAL's lack of error trapping, this makes much of my programming unstable.

KYAN does offer a systems utilities disk that offered such things as disk directories, but as the price was approximately $50.00, which is what I spent for KYAN PASCAL in the first place, I couldn't bring myself to lay out that much money for it (especially with rumors of an upcoming update that would take care of many of the forenamed problems).

* ODDS NDS *

In addition to the poor manual and "DOS-lesness", there were a few other problems that hindered programming with KYAN PASCAL, such as:

1. KYAN PASCAL would only run on an 800XL or 130XE. If you had an unmodified 800, then you were out of luck (although an upcoming 48K version was promised).

2. Identifiers could not include an underscore character. The variable RATE_OF_PAY is much more readable than RATEOFPAY.

3. KYAN PASCAL booted from a modified DOS 2.5, therefore you couldn't use a more powerful DOS, or use true double-density storage.

4. PEEKs and POKEs were excluded.

5. The GOTOXY (cursor positioning) procedure was left out. This made screen formatting not fun.

With the exception of the problems and shortcomings mentioned, KYAN PASCAL was an exceptional value for $69.95 (less through discounters). A few other PASCALs were available for the 8-bit Atari, but KYAN PASCAL was (and still is) the Imperial Battlestar of PASCAL compilers for the 8-bitters.

Chapter Two
KYAN PASCAL STRIKES BACK

A while ago (quite a while - this article missed deadlines), I received notice that KYAN PASCAL was now updated to Version 2.01A, and for $24.95 ($20 for the update and $4.95 shipping), and the return of the original V.1.1 disk, I could receive my update package.

Although at first I thought that $24.95 was quite high for an update, I sent my money and disk. A couple of weeks later, my trepidation was eased somewhat when I received a new, complete KYAN PASCAL package. Instead of copying the update onto the old disk, or just sending a new disk, they sent a whole, brand new package, just as if I had ordered a new one.

Let's take a look at what KYAN did (and didn't do) to improve their PASCA

* THE MANUAL II *

The new manual is a definite improvement over the old, in many ways!

First of all, it is now 300 pages and is bound in an attractive, high-quality three ring binder, and is now ATARI-SPECIFIC! The editing and compiling instructions are much better written.

Each chapter begins with an OVERVIEW which introduces the upcoming concepts. Following is THE PROGRAM, which is built upon the related concepts. THE LOGIC OF THE PROGRAM is a step by step explanation of the program. Then comes a few GENERAL COMMENTS followed by TOPICS FOR PROGRAMMERS and ADVANCED TOPICS (or TECHNIQUES), which introduce the more refined of PASCAL commands. Last is the CONCLUSION which sums up the previous chapter and introduces the next one.

The example programs supplied are much more suitable for learning, and do not get too complex too soon.

Although the concepts of PASCAL programming are better explained, SAM'S PASCAL PRIMER, or another good PASCAL book is still a must.

The manual's coverage of including ASSEMBLY LANGUAGE within your PASCAL source code is greatly expanded, although it is not meant as a tutorial. A good deal of A.L. programming experience is required. NOTE: a few changes in A.L. code must be made if you wish to re-compile

(continued on page 11)

code from V.1.1 on V.2.01A, although the changes are minor.

The list of error messages has been expanded from 35 to 141 reportable errors.

All in all, the new manual is well written and well laid out. The quality is first rate. It almost justifies the high update price. The amazing thing is that the $69.95 retail price was not affected.

* NEW FEATURES/CHANGES

KYAN PASCAL now includes KIX, which allows you access to DOS commands from the PASCAL programming environment, but unfortunately not from the program itself (looks like I'll need to buy the utility disk after all).

KIX is actually a series of files which can be called from the system prompt (which has been changed to % as opposed to the older >).

Following are the KIX commands and what they do:

PWD - (Print Working Device) Prints the number of the current default device.

CD - (Change Device) Changes default drive.

LS - (List Directory) Prints directory to screen, printer, or a file.

CAT - (Concatenate) Appends one or more files to a new file.

CP - (Copy) Copies files - use to move files to Ramdisk.

MV - (Move) Renames files.

RM - (Remove) Deletes files.

CHMOD - (Change protection Mode) Locks or unlocks files.

FORMAT - (Format disk) Formats a disk in either single or enhanced density.

SD - (Screen Dump) Copies the screen contents to the printer.

While working with KIX, you need not specify device numbers. KIX will automatically search first the Ramdisk, then the user device, and then the default. KIX will also accept both upper and lower case, converting all to upper case.

The only thing lacking is the ability to batch process KIX commands. It would make setting up your Ramdisk much easier, for instance.

* ODDS NDS II *

A few of the minor complaints have been cleared up in Version 2.01A.

1. KYAN PASCAL V.2.01A will now run on any 48K Atari.
2. Underscores are now accepted.
3. It will now boot from most DOSes.
4. Two ASSEMBLY LANGUAGE routines are included to allow PEEKs and POKEs.
5. Unfortunately, GOTOXY is still NOT supported, but the manual explains how to use the POSITION graphics procedure to position the cursor.

Kyan Pascal will be continued next month so stay tuned right here on the RI ACE Reporter for Kyan Strikes Back Part II by a Reasonably Informed source.
Ed.

The Software Center of R.I.

2105 Post Road Warwick R.I.

across from the Airport

software for all atari computers

epson printer specialist's

call 738-9800

OPEN MONDAY - SAT.
9AM - 5PM

BEST BACKUP OUT FOR ST
ed.

ProCopy ST BACKUP UTILITY

You can't backup your software because copy protection locks you out. **ProCopy** is the key!

- Protects against the accidental loss of expensive software Version 3.2
- Works with all Atari STs
- Copies both S/S & D/S disks
- Use with 1 or 2 disk drives
- Not copy protected
- **FREE** shipping and handling
- **TOLL-FREE** telephone number
- Updates are only $12 with return
- Orders shipped same day
- Disk analyze function included

Dealer Inquiries Welcome

VISA and C.O.D. orders

Call (800) 843-1223

$34.95

Send check for $34.95 (overseas add $2.00 for air mail) to:

PROCO PRODUCTS
P.O. BOX 665, CHEPACHET, RHODE ISLAND 02814 USA
(401) 568-8459

Available Internationally In Europe and Australasia

THE MOVING FINGER CO.
Building 2
Shamrock Quay
Southampton, SO1-1QL
England
Tel. 0703-227274

TECH-SOFT COMPUTER WHOLESALE
324 Stirlir Highway
Claremont
Western Australia 6010
Tel. 09-385-1885

OCAOC

OF DOUBLE DENSITY

By Chuck Hosick, ACAOC

Many of the new disc drives available for the ATARI are double density and the venerable 810 can be made double density with the NCT Turbo modification. Double Density means that instead of the 128 bytes of data in each sector there are 256. Practically, this means 253 vis 125 because of the DOS overhead of 3 bytes per sector. For DOS files, double density more than doubles the capacity. Most drives sense the density of the disc that you boot with and switch densities accordingly. But what if you want to change horses in the middle of the stream? i.e. change densities from within a BASIC program. This can be done easily with a routine that calls on the Operating System's SIO program. I'm not going to give a tutorial on SIO because assembly language programmers should already know about it and BASIC programmers don't even need to know how to spell it. (The interested reader is directed to the Technical User's Notes and De Re Atari.)

Double Density drives have two additional SIO commands that they respond to. These are READ and WRITE OPTION TABLE. The Option Table is a twelve byte table which contains information about the configuration of the drive. The bytes and what they mean are:

Byte 0 - Number of tracks, normally 40
Byte 1 - Head Step Rate
- 0 - 6ms
- 1 - 12ms
- 2 - 29ms
- 3 - 30ms

Byte 2&3 Number of sectors/track, normally 18
Byte 4 - Number of sides, 0 for 1
Byte 5 - Density, 0 for single, 4 for double
Byte 6&7 Number of bytes/sector
Byte 8 - Drive present
- 0 for not on line
- 1 for on line

Byte 9 - ACIA init bits
Byte 10&11 future use

The protocol, established by Percom, is to first read the table, make changes accordingly, and write it back to the drive. Listing 1 is a BASIC program which you can use as a guide in including a change density function in a BASIC program.

Both the drive and DOS must be changed. DOS 2.0 is capable of communicating with either single or double density drives. If it is not patched for double density, you have to limit the number of disc files that you have open - but that is not normally a problem. DOS 2.0 maintains a table, starting at $1311(4881), which contains a density code for each drive. The code is a 1 for single density and a 2 for double. Whenever the density is changed this table must be changed as well. Other DOSs have similar tables, but they may not be in the same place.

The BASIC program in Listing 1 sets up a string to accept the Option table from the drive. Lines 25 thru 60 set up the variables for READing from drive 1. Change the number in line 25 for a different drive number. Lines 70 thru 170 set up the Device Control Block (DCB) for the call to SIO. The call to SIO is contained in the USR function in line 180. This is a simple imbedded machine language subroutine. This is necessary because the operating system does not provide an entry point from BASIC. The USR function pushes the number of variables onto the stack on top of the return address. Therefore, a PLA must be executed before the call to SIO or at least before the return to BASIC can be executed. (It would have been easy for ATARI to have included a PLA before its jump to accomodate BASIC - but that's life.) The rest of the program is a demonstration of actually changing the density. Lines 200 and 210 check the present density and branch to the proper change routine. The change routines change both the drive and DOS. The drive is changed by changing the density byte and the SIO control parameters to write and then looping back to set up the DCB and call SIO. DOS is changed by POKEing the new density value into the Drive Table.

The concepts presented in this program can be used in any program. Remember to set up the DCB as shown in lines 70 thru 170, call SIO as in line 180, change the density, change the READ DCB bytes to WRITE. call SIO again, and then POKE DOS and that's all there is to that song.

```
0 REM SAVE"D:DENSCONT.SAV
10 REM Chuck Hosick, A.C.A.O.C.
20 DIM OPTAB$(12),ANS$(3),CMND$(1)
25 DRIVNO=1
30 OPTAB$="            "
35 CMND$="R":REM R FOR READ,
                W FOR WRITE
40 RDTAB=4*16
50 WRTAB=8*16
55 DIREC=RDTAB:REM OR WRTAB
60 DCB=3*256
70 POKE DCB,49:REM DRIVE ID
80 POKE DCB+1,DRIVNO:REM ;
   NO. OF DRIVE TO CONTROL
90 POKE DCB+2,ASC(CMND$)
100 POKE DCB+3,DIREC
110 ADTAB=ADR(OPTAB$)
120 HI=INT(ADTAB/256)
130 LO=ADTAB-(HI*256)
140 POKE DCB+4,LO:REM BUFFER SPACE
150 POKE DCB+5,HI
160 POKE DCB+6,12:REM ;
    OPTAB$ IS 12 BYTES LONG
170 POKE DCB+9,0:REM JUST IN CASE
180 X=USR(ADR("h Y[d+"));REM ;
    PLA,JSR SIO,RTS
    REM L/C H,SPACE,U/C Y,
    INVERSE L/C D,CONTROL PERIOD
200 IF OPTAB$(6,6)="♥" THEN 250:REM ;
    ♥ MEANS SINGLE DENSITY
210 IF OPTAB$(6,6)="┤" THEN 350:REM ;
    ┤ MEANS DOUBLE DENSITY
220 ? "ERROR IN READING PARAMETERS"
230 END
240 REM DRIVE IS SINGLE DENSITY
250 ? "DRIVE NO. ";DRIVNO;
255 ? " IS SINGLE DENSITY"
260 ? "DO YOU WANT TO CHANGE IT";
265 ? " - Y/N ";
270 INPUT ANS$
280 IF ANS$(1,1)="Y" THEN 300
290 END
300 OPTAB$(6,6)="┤"
310 DIREC=WRTAB
320 CMND$="O"
330 POKE 19*256+16+DRIVNO,2:REM ;
    SETS DOS TO D.D.
340 GOTO 70:END
345 REM DRIVE IS DOUBLE DENSITY
350 ? "DRIVE NO. ";DRIVNO;
355 ? " IS DOUBLE DENSITY"
360 ? "DO YOU WANT TO CHANGE IT ";
365 ? " - Y/N ";
370 INPUT ANS$
380 IF ANS$(1,1)="Y" THEN 400
390 END
400 OPTAB$(6,6)="♥"
410 DIREC=WRTAB
420 CMND$="O"
430 POKE 19*256+16+DRIVNO,1:REM ;
    SETS DOS TO S.D.
440 GOTO 70:END
```

CATCHING THE MOUSE

AACC FEEDBACK 12/1 - 87/88

(36)

by Gordon Billingsley

Reprinted from Aetun, Utah, August 1987

One of the first things most BASIC programmers noticed about ST BASIC was that there was no command for using the mouse. Fortunately, that's not the whole story. The peeks and pokes that give BASIC access to the GEM interface also give programmers access to the mouse.

It can be used to point, to select (with either button) or to provide co-ordinates for invoking other features one easy gosub call. GOSUB GETMOUSE -- as set up here -- tells you where the mouse pointer is located and whether one of the mouse buttons has been depressed.

Here is the subroutine for using the mouse:

```
10 GETMOUSE:poke control, 124
20 poke contrl+2,0:poke contrl+6,0:vdisys(0)
30 mx=peek(ptsout):my=peek(ptsout+2)
40 button=peek(intout):RETURN
```

The xy co-ordinates are returned in the variables mx and my. Keep in mind these are screen-oriented co-ordinates and not BASIC window-oriented co-ordinates -- that is, co-ordinated 0,0 is in the upper left corner of the screen, which is outside the BASIC window. Co-ordinates for most BASIC statements start within the output window.

Mouse button presses may be detected in a variable call button, with values as follows:

0=no button press; 1=left button pressed; 2=right button pressed.

Here's a short routine to demonstrate the use of GOSUB GETMOUSE

```
10 gotoxy 2,5:?"Click left button to stop"
20 GOSUB GETMOUSE
30 gotoxy 2,2:?"X=";mx;"Y=";my
40 if button =1 then 60
50 goto 20
60 end
```

Notice line 50. It creates an endless loop that forces the program to cycle through GOSUB GETMOUSE repeatedly until a condition that causes it to branch outside the loop is met. This is the sort of routine that can be used for menu selections. Your program won't be doing anything else while waiting for you to make a menu selection.

This little routine also is a handy programming utility. You may place menu items, markers or any other item on the screen then use this routine to identify its exact positions and boundaries. That comes in handy when you want the pointer to be within a specific range of pixels for a button press to be valid. Here's

an example from a game I wrote in which you could choose the number of players to be involved.

```
280 gotoxy 4,6:?"How many players?"
300 HEAD$=CHR$(28)+CHR$(29):chin$=Chr$(30)+chr$(31)
320 GOTOXY 15,6:?head$:gotoxy 15,7:?CHIN$
340 for !=1 to 2:gotoxy 14+1,8:?head$:gotoxy 14+1,9:?chin$:next !
380 for !=1 to 3:gotoxy 14+1,10:?head$:gotoxy 14+1,11:?chin$:next
!
420 for !=1 to 4:gotoxy 14+1,12:?head$:gotoxy 14+1,13:?chin$:next
!
480 GOSUB GETMOUSE
500 if(my>145)*(my<170)*(mx>250)*(mx<275)*(button=1) then players
=1:goto 620
520 if(my>180)*(my<205)*(button=1)*(mx>250)*(MX<290) then
players=2:goto 620
540 if(my>215)*(My<240)*(mx>250)*(mx<310)*(button=1) then players=
3:goto 620
560 if(my>247)*(my<275)*(mx>250)*(mx<325)*(button=1) then players=
4:goto 620
580 if button=1 then gotoxy 0,0:?chr$(7)
600 goto 480
620 END
```

This routine uses the built in graphic face of ST BASIC as a graphic symbol for the number of players in a game. The mouse pointer and button will select one of the lines of the faces only if it is in the vicinity of the faces. Line 580 causes the internal bell to ring if you press the mouse button in a location away from the faces. A hint on using the print bell function: use the gotoxy command just before it. If you don't it will scroll the screen up one line as if you were printing text without a gotoxy command.

Those are the basics - if you'll excuse the pun - of using the mouse. In the next column, we'll look at ways to jazz up your use of the mouse.

Software Exchange Wanted

I have a copy of Alpha Systems- Atari Software Protection Techniques with disk, in brand new condition, which I wish to exchange for the sequel- Alpha Systems Atari Advanced Software Protection Techniques with disk. I may be contacted at home on (049) 51-5731 or at work on (049) 66-9317. Ask for Bill Hanna.

BAXTER AND DUNBAR." You didn't believe them for a minute, did you?

Due to the dynamic nature of Deadline, there are several ways to end up accusing Baxter and Dunbar of the murder. There are also more puzzles to solve, but this is all that is necessary to put together an air-tight case against them.

CONGRATULATIONS! You have just solved the mystery of DEADLINE!

DOS 2.5

kc ACE July 85

Reviewed by Steve Fabac

The following is a review from the introduction to DOS 2.5 received from ATARI along with the DOS 2.5 disk. It appears from reading the booklet, that you will have to order a more complete DOS 2.5 manual directly from Atari for $10 plus $2.50 for shipping and handling.

As received DOS 2.5 is in ehanced density and so you must have a 1050 drive to start using it. Once the disk is booted, you can use the new format command P (FORMAT SINGLE) to format a disk in single density using the 1050 drive. Then you can use the H (WRITE DOS FILES) to write DOS 2.5 in single density.

Contained on the DOS 2.5 disk is a file named RAMDISK.COM that automatically sets up the extra 64K of RAM in the 130XE as a RAM DISK. So when you boot DOS 2.5 on the 130XE, DOS will inform you that it is initializing the RAM DISK. It will set the extra 64K up as D8. And establish DUP.SYS and MEM.SAV on the RAM DISK so that the RAM DISK copies will be accessed (at high speed) rather than the floppy disk versions.

One caution, you can use all DOS commands with the RAM DISK EXCEPT J - DUPLICATE DISK to the RAM DISK. The RAM DISK has room only for 499 sectors. Therefor you can't dup either a single or enhanced density disk. Instead you must use the copy file command and take care that the file does not exceed the capacity of the RAM disk. Atari further cautions that if you dup the RAM DISK to a floppy, that floppy will be capable of accessing only 499 sectors.

There are three new utility programs provided on the DOS 2.5 disk: COPY32.COM to copy files from DOS 3.0 to DOS 2.5. DISKFIX.COM allows you to "correct some problems that may occur with files on DOS 2.5 and DOS 2.0S disketts." Atari also states "under certain conditions, you can also use this utility to recover deleted files." (How's that for not being specific? Sounds like they don't know what problems may occur or under what conditions you can un-delete files.) And SETUP.COM to allow you to change certain (again unspecified) DOS parameters.

Luckely, COPY32.COM, DISKFIX.COM, and SETUP.COM prompt the user with brief instruction for their use. For more detailed instructions, you will have the order the DOS 2.5 manual from ATARI.

kc - ACE July 85

Disk Density Control

by Roy Nickum

Have you ever wondered about the operation of a multiple density capable disk drive (any drive but the Atari™ 810)? These drives all contain a twelve byte OPTION table in RAM (in the drive, not in the computer) which defines the current drive density. The table is initially configured by the default switches on the drive when it is turned on. The next entry into the table occurs when a "boot" disk is loaded into the computer. The microprocessor in the disk drive is programmed to sense the disk density and reconfigure the table.

After the boot process, then what? Well, with DOS XL™ or SmartDOS™ or some of the other DOSes the reconfigure command will change the drive density. But what about control of the drive

density from within a program? It's not only possible, it's not even hard.

To that end Table 1 is a description of the twelve byte option table.

Table 1, Drive Configuration

Note: All 2-byte values are in Most Significant Byte, Least Significant Byte format (the opposite of normal 6502 practice).

BYTE	Function
0	Number of tracks (literal value)
1	Head step rate: Value — Rate 00 — 30 msec. 01 — 20 msec. 02 — 12 msec. 03 — 06 msec.
2,3	Sectors/Track (literal value) 18 = Single or double density 26 = Atari™ dual (enhanced) density
4	Number of sides (Heads) 0 = 1 side 1 = 2 sides
5	Density 00 = Single density 04 = Double or Atari™ dual (enhanced) density
6,7	Sector size (literal value) 0,128 = Single or Atari™ dual (enhanced) density 1,0 = Double density
8	Drive present 00 = Drive not present Any other value = Drive present
9,10,11	Future options

These drives allow access to the option table by adding 2 new commands to the 5 commands the disk drive normally understands. Command number 78 reads the option table while number 79 writes it. However, some Atari™ 810's recognize the write option table command exactly like a write sector command. Use this command with caution.

Listing 1 is a short BASIC program which will read and write the table for the selected drive. Lines 140-190 set up two machine language subroutines which will be described later. Lines 200-290 set up the Device Control Block (DCB) for the disk read operation while lines 300-310 request the drive number and place it in the DCB.

Line 320 uses the first of the machine language subroutines to have the operating system read the selected drive's option table. Lines 330-460 display the current table entries. Lines 470-670 request the new density and configure the DCB for a disk write operation to the table. Line 680 uses the first machine language subroutine again, this time to write the table.

The drive has now been configured to a new density. However, one important task remains. If left in this state the drive is operating in one density while the DOS is expecting another density. This can lead to some confusing results. Yes, the DOS also keeps track of the density. Even Atari DOS 2.0™. DOS sets locations in the computer's RAM to indicate the density and allocates buffers which are sized to hold the disk information before and during transfers. Eventhough the programs will configure the 1050 to enhanced density, you must be using Atari DOS 2.5™ or you will not be able to access the 1050 drive. The second machine language subroutine (called from Line 690) calls the DOS initialization routine. This will cause DOS to inquire about the current drive configuration and configure itself accordingly.

Listing 1, Density Control

```
100 REM *** DISK DENSITY CONTROL ***
110 REM ***     SAMPLE PROGRAM     ***
115 REM ***     by Roy Nickum      ***
120 DIM BUF$(12): REM OPTION TABLE
130 DIM R$(1): REM RESPONSES GO HERE
140 Z=ADR("    "):Z1=ADR("    "): REM
    JUMP TO DISK I/O; JUMP TO DOS INIT
150 DATA 104,76,89,228,104,76,224,7
160 FOR I=0 TO 3
170 READ X:POKE Z+I,X:NEXT I
```

```
180 FOR I=0 TO 3
190 READ X:POKE Z1+I,X:NEXT I
200 DCB=768
210 DATA 49,1,78,64,0,0,1,0,12,0,0,0
211 REM DEVICE, UNIT, COMMAND,
    STATUS(DIR), BUFFER ADDR, TIMEOUT,
    UNUSED, # BYTES, SECTOR #
220 FOR I=0 TO 11: REM BUILD THE DCB
230 READ X
240 POKE DCB+I,X
250 NEXT I
260 A=ADR(BUF$): REM INSERT BUFFER
    ADDRESS
270 X=INT(A/256)
280 POKE DCB+5,X
290 POKE DCB+4,A-X*256
300 PRINT "DRIVE NUMBER"; :INPUT I
310 POKE DCB+1,I: REM SET DRIVE IN DCB
320 X=USR(Z): REM READ THE OPTION TABLE
330 PRINT "DRIVE NUMBER OF TRACKS = ";
    PEEK(A)
340 ON PEEK(A+1) GOTO 360,370,380
350 X=30:GOTO 390
360 X=20:GOTO 390
370 X=12:GOTO 390
380 X=6
390 PRINT "DRIVE STEP RATE = "; X; "
    MSECS."
400 PRINT "DRIVE SECTORS/TRACK = ";
    PEEK(A+3)
410 PRINT PEEK(A+4)+1; " SIDED DRIVE"
420 IF (PEEK(A+5)=0 AND PEEK(A+3)=18)
    THEN PRINT "SINGLE DENSITY":GOTO 450
430 IF (PEEK(A+5)=4 AND PEEK(A+3)=18)
    THEN PRINT "DOUBLE DENSITY":GOTO 450
440 IF (PEEK(A+5)=4 AND PEEK(A+3)=26)
    THEN PRINT "ENHANCED DENSITY"
450 PRINT PEEK(A+6)*256+PEEK(A+7); "
    BYTES/SECTOR"
460 PRINT "Make it single (S), enhanced
    (E),":PRINT "or double (D) density";
470 INPUT R$
480 IF ASC(R$)=ASC("D") THEN 570
490 IF ASC(R$)=ASC("E") THEN 620
500 IF ASC(R$)=ASC("S") THEN 520
510 GOTO 460
520 POKE A+5,0: REM MAKE SINGLE DENSITY
530 POKE A+3,18
540 POKE A+6,0
550 POKE A+7,128
560 GOTO 660
570 POKE A+5,4: REM MAKE DOUBLE DENSITY
580 POKE A+3,18
590 POKE A+6,1
600 POKE A+7,0
610 GOTO 660
620 POKE A+5,4: REM MAKE ENHANCED DEN-
    SITY
630 POKE A+3,26
640 POKE A+6,0
650 POKE A+7,128
660 POKE DCB+3,128: REM SET DIRECTION
    FOR WRITE
670 POKE DCB+2,79: REM SET COMMAND
    (WRITE)
680 X=USR(Z): REM UPDATE DRIVE OPTIONS
690 X=USR(Z1): REM UPDATE DOS
700 END
```

For those who want the same functionality without the need for interaction, Listing 2 will set up a machine language subroutine which will, depending on how it is called, read the table or read and write the table. If called as:

```
X=USR(ADR(D$),#D)
```

The option table will be read and "X" will be returned as:

```
CHR$(X)="S" for single density
        "E" for Atari™ dual
            (enhanced) density
        "D" for double density.
```

If called as:

```
X=USR(ADR($D),#D,Density)
```

the option table will be read then written with the new density. This is followed by a reinitialization of DOS (to keep communication straight). "X" will be returned with the density prior to reconfiguration.

Now you have the last word in the control of the drive's density.

Listing 2, "USR" Density Control

```
10 REM DRIVE DENSITY USR FUNCTION
20 REM by Roy Nickum; 20 June, 1895
30 REM TO READ DENSITY:
31 REM X=USR(ADR(D$),#D)
32 REM #D=DRIVE NUMBER
33 REM RETURNS X AS:
34 REM ASC(DENSITY)
```

```
35 REM DENSITY=
36 REM "S" FOR SINGLE
37 REM "E" FOR ENHANCED 1050
38 REM "D" FOR DOUBLE
40 REM TO WRITE DENSITY:
41 REM X=USR(ADR(D$),#D,DENSITY)
42 REM #D & DENSITY AS ABOVE
43 REM RETURNS X AS DENSITY PRIOR
44 REM TO RECONFIGURATION
50 DIM D$(178)
60 FOR I=1 TO 178
70 READ DATA
80 POKE ADR(D$)+I-1,DATA
90 NEXT I
1000 DATA 104,133,203,104,104,141,1,3,
     169,49
1010 DATA 141,0,3,169,78,141,2,3,169,64
1020 DATA 141,3,3,169,0,141,4,3,169,4
1030 DATA 141,5,3,169,1,141,6,3,169,12
1040 DATA 141,8,3,169,0,141,9,3,32,89
1050 DATA 228,173,5,4,201,0,208,4,169,83
1060 DATA 80,24,173,6,4,201,1,208,4,169
1070 DATA 68,80,13,173,3,4,201,26,208,4
1080 DATA 169,69,80,2,169,85,133,212,
     169,0
1090 DATA 133,213,165,203,201,2,240,1,
     96,169
1100 DATA 79,141,2,3,169,128,141,3,3,141
1110 DATA 179,6,169,0,141,5,4,141,6,4
1120 DATA 141,10,3,141,11,3,169,18,141,3
1130 DATA 4,104,104,201,83,240,34,201,
     68,208
1140 DATA 16,162,0,142,7,4,232,142,6,4
1150 DATA 162,4,142,5,4,80,14,201,69,208
1160 DATA 16,162,4,142,5,4,162,26,142,3
1170 DATA 4,32,89,228,32,224,7,96
```

The following is the machine language source code for the data statements in listing 2.

```
2       *=  $0400     ;ASSIGN
    BUFFER=$400
4   BUFFER
10      *=  $0600     ;ORG FOR ASSEMBLY
20      PLA           ;GET NUMBER ARG
30      STA 203       ;SAVE NO. OF ARG
40      PLA
50      PLA           ;LO BYTE OF ARG
60      STA 769       ;SAVE DRIVE NUMBER
70      LDA #49       ;SET AS DISK
80      STA 768       ;DEVICE CODE
90      LDA #78       ;SET READ CONFG.
0100    STA 770       ;DCB COMMAND
0110    LDA #64       ;DIRECTION = READ
0120    STA 771
0130    LDA # <BUFFER;LO-BYTE
0140    STA 772
0150    LDA # >BUFFER;HI-BYTE
0160    STA 773       ;POINTER TO BUFF
0170    LDA #1        ;SHORT TIMEOUT
0180    STA 774
0190    LDA #12       ;BUFFER = 12 BYTE
0200    STA 776
0210    LDA #0
0220    STA 777       ;LENGTH OF BUFFER
0230    JSR $E459     ;USE OS TO READ
                      ;DRIVE CONFG.
0240    LDA BUFFER+5  ;DENSITY STATUS
0250    CMP #0        ;IS DENSITY SNGL?
0260    BNE DOUBLE    ;NO, IS DOUBLE
0270    LDA #83       ;YES, "S" FOR
0280    BVC WRITE.TEST ;RETURN TO BASIC
0290 DOUBLE
0300    LDA BUFFER+6
0310    CMP #1        ;IS IT DOUBLE?
0320    BNE ENHANCED  ;NO
0330    LDA #68       ;YES, SET "D"
0340    BVC WRITE.TEST
0350 ENHANCED
0360    LDA BUFFER+3
0370    CMP #26       ;IS IT ENHANCED?
0380    BNE UNKNOWN   ;NO
0390    LDA #69       ;YES, "E"
0400    BVC WRITE.TEST
0410 UNKNOWN          ;UNKNOWN RESPONSE
0420    LDA #85       ;"U" TYPE
0430 WRITE.TEST
0440    STA 212       ;LO-BYTE
0450    LDA #0
0460    STA 213       ;HI-BYTE
0470    LDA 203       ;CHECK NO. ARG
0480    CMP #2        ;WRITE REQUEST?
0490    BEQ WRITE     ;YES, DO WRITE
0500 EXIT             ;NO, SO RETURN TO
0510    RTS           ;BASIC
0520 WRITE
0530    LDA #79       ;SET WRITE CONFG.
0540    STA 770
0550    LDA #128      ;CMD = WRITE
0560    STA 771
0570    STA BUFFER+7  ;SET SINGLE DNSTY
0580    LDA #0        ;AS DEFAULT MODE
0590    STA BUFFER+5
0600    STA BUFFER+6
0610    STA 778
0620    STA 779
0630    LDA #18
0640    STA BUFFER+3
0650    PLA           ;GET DENSITY FROM
0660    PLA           ;STACK NEXT
0670    CMP #83       ;"S"?
```

```
0680      BEQ WRITEIT  ;YES, WRITE SNGL
0690      CMP #68      ;"D"?
0700      BNE ENHANCED?;NO, CHECK IF "E"
0710      LDX #0       ;YES, SET DOUBLE
0720      STX BUFFER+7 ;DENSITY CONFIG
0730      INX
0740      STX BUFFER+6
0750      LDX #4
0760      STX BUFFER+5
0770      BVC WRITEIT  ;DO IT
0780 ENHANCED?
0790      CMP #69      ;IS IT "E"
0800      BNE EXIT2    ;NO, NOT E
0810      LDX #4       ;SET ENHANCED
0820      STX BUFFER+5 ;DENSITY MODE
0830      LDX #26
0840      STX BUFFER+3
0850 WRITEIT
0860      JSR $E459    ;RE CONFIG DRIVE
0870      JSR $07E0    ;RE-INITIALIZE DOS
0880 EXIT2             ;TO NEW CONFIG
0890      RTS
```

FOR SALE
Call Larry Jordan
765-4627

Atari CX85 Numeric Key Pads. New in boxes with schematics, documentation and device handlers.

$15.00

IN MY OPINION

By Larry Jordan

Some members of our group who have purchased SWP's ATR-8000 interface and CPM compatible computer have recently conveyed to me a surprising lack of support from SWP in that when they call SWP long distance to either ask questions or point out problems they get little if any aid for their problems. SWP either pleads ignorance of the software that it promotes and sells or in two particular cases merely hangs up on you.

Such support for the customer would certainly not indicate to me that SWP has any long range plans to continue to try and sell their product to the ATARI using public. One specific case in point is the promotion thru SWP's newsletter of a "new" version of My-Dos which they are actively trying to sell when indeed there is in fact a more advanced version available in the software stores for the same price. when SWP was asked about this they replied that they had no knowledge of the other version. When the point was made about they're being My-Dos's largest customer - who should be able to wield some influence with My-Dos - they simply hung up the phone. Is this the response one would expect from a company that you have spent hundreds of dollars with?

I for one would urge that SWP examine its customer support attitude and make any needed changes forthwith. It would seem to me that any time one asks a question about using their product they claim ignorance of the software which they require you to use to utilize their product - kind of a catch 22. If they are indeed so ignorant of this software, why continue to sell it?

While I have not been praising SWP thus far let me add that what makes this a truly sad situation is that they make - as far as I am concerned - probably the most useful product available for the Atari computer user. I myself bought a 16 K version in lieu of an Atari 850 interface to enable me to use a printer and modem. However, without the My-Dos RS-232 handler it was of no use in using my modem. Additionaly, when I purchased my unit - which is no longer available - no dos was supplied with it. Now one must buy the 64K unit and it does come with the software.

In closing let me say that all in all I feel that SWP makes a great product but needs to examine its customer relations policy. The idea of tying themselves to a particular dos manufacturer seems to have put them at My-Dos's mercy. The only other dos which to my knowledge supports the ATR user is Sparta Dos and it is not compatible with any other dos's. If anyone knows of any other dos that supports the ATR RS-232 port please let me know.

ICEBERG - REACT 3-86 (12)

BASIC DYNAMIC DENSITY CHANGER

by Howard Forsberg

After reading the "INSIGHT:Atari" series of articles on the SIO call and Percom Config Block, I quickly typed in the Config Block Modifier. I knew that this was finally the answer to my problems of having to reboot my system every time I wanted to access both single and double density disks with the same BASIC program. But, alas, my hopes for a solution had only partially been realized. The program could still only process one density at a time dynamically. To change densities still required the pressing of the "RESET" key in order for the change to take effect.

This was better than rebooting with the system, but still wasn't making the change dynamically. I knew that machine language programs could make the change, but I wanted this to be done from BASIC.

One night I decided to determine exactly what had to be done in order to make the density change totally dynamic within a BASIC program. I got out my "Inside Atari DOS", "Mapping the Atari", and the series of articles on SIO from "Compute" magazine and went to work. After about two and a half hours of reading, trying, peeking, and poking, I had determined there were about 5 or 6 different places that were referenced during disk I/O that had to do with density.

After booting the system in single density, I ran the Config Block Change routine to set the disk up for double density. Then, I poked in the values I had decided would make the system access the disk as double density. When I tried to list the program to a preformatted double density disk, it worked. Now all I had to do is to decide what really needed to be done, and what the system would handle for me.

After a little more trial and error, I came up with just two pokes that needed to be made. The first was at dec. 768, which was in the command string for the config block change. After returning from the change, it needed to be poked with a "1". The other was in a byte called DRVTBL, which is set up by the DOS during initialization. The location is dec. 4881 for drive 1. The value is either a "1" or "2" for single or double density. That was all that was required, everything else was changed by the system.

Well, I was feeling kind of proud of myself, but was starting to wonder about "Why" this was happening. I still didn't know if it was a fluke, or something I could rely on happening every time. So it was back to the books. After about 6 or 7 more hours of reading, I believe I know why. This is a brief explanation of what is happening.

During the "boot" process, boot sector "1" is copied into memory starting at $700. At address $70A is the DRVBYT, which identifies the drive as a "1" or "2" for single or double density. FMS initialization copies the DRVBYT to the DRVTBL displacement for the drive number being used. The Atari DOS uses a routine called "SETUP" to set up the FMS control cells for all commands to access disk files (including disk initialization). The setup routine copies the Drive Type into the DRVTYP byte from the DRVTBL displacement for that drive number. The setup routine then sets up all the fields needed to access the disk based on the DRVTYP value. By poking the new value for the Drive type into the DRVTBL displacement before executing any I/O commands, DOS sets up the new values during the next pass through the "SETUP" routine.

I have used this process for cataloging my disks, and creating labels for them. My BASIC programs both automatically change densities and continue processing by using this routine. I have condensed the routine as much possible, because my catalog program keeps the entire disk library in memory. The addition of this routine only caused a loss of 4% of the total number of records available. The routine will read the config block and change to the other density every time it is called.

The breakdown of the routine is like this. Line one dimensions the strings and initializes them for nonvariable values. DRV$ is the device number and can be changed if other drives are to be accessed. If another drive is used, the DRVTBL displacement also changes. The poke addresses on line 29300 must reflect the new displacement (ex. drive 3 would be address 4883). Line 29800 is the first call that reads the current configuration for the device. If an error code greater than 127 occurs than the program stops. The problem with the drive was not density. The next two lines switch the density from the one read. Line 29200 sets up the write and when returned to, also stops if the error was not a density problem. Line 29300 pokes the required values to into 768 and DRVTBL so the change will take affect without a "RESET" and returns to the basic program.

To use the routine, add line one to the front of your program. Set a trap to 29800 before you access the disk. You may want to Peek 195 for a error 144 before continuing. When a disk error occurs, the routine will reset the density for both the drive and the system. You will need to close the IOCB on line 29310 before returning to the line that does the open.

One other fact that needs to be mentioned is that this routine will work when it is loaded with Atari DOS. I have tried it when loaded from MYDOS, and it does not work. I haven't tried it with any other versions, so I don't know if they will work or not. I feel like the reason for this problem is that the other versions have moved the displacement for the DRVTBL.

(13)

BASIC DYNAMIC DENSITY CHANGER
by Howard Forsberg.

This routine allows the DYNAMIC changing of density within a BASIC progras. There is no need to press "RESET" after the drive is reconfigured.

Line one dimensions the strings and initializes thes for nonvariable values. DRV$ is the device nusber and can be changed if other drives are to be accessed. If another drive is used, the pokes on line 29300 need to reflect the new displacesent of DRVTBL. For exasple, if drive 3 were used, the poke address would be 4883. Line 29000 is the first call that reads the current configuration for the device. If an error code greater than 127 occurs than the progras stops. The problss with the drive was not density. The next two lines switch the density fros the one read. Line 29200 sets up the write and when returned to, also stops if the error was not a density problss. Line 29300 pokes the required values to into 768 and DRVTBL so the change will take affect without a "RESET" and returns to the basic progras.

To use the routine, add line one to the front of your progras. Set a trap to 29000 before you access the disk. You say want to Peek 195 for a error 144 before continuing. When a disk error occurs, the routine will reset the density for both the drive and the systss. You will need to close the IOCB on line 29310 before returning to the line that does the open.

One other fact that needs to be sentioned is that this routine will work when it is loaded with Atari DOS. I have tried it when loaded fros MYDOS, and it does not work. I haven't tried it with any other versions, so I don't know if they will work or not. I feel like the reason for this problss is that the other versions have soved the displacesent for the DRVTBL.

```
1 DIM TBL$(12),CMD$(1),CAL$(14):CAL$="h Y[illegible]":DRV=1
29000 TBL=ADR(TBL$):ADDR=TBL:CMD$="N":GOSUB 29500:DEN=PEEK(T
BL+5):IF SST>127 THEN STOP
29100 IF DEN=4 THEN DEN=0:BPS=128:GOTO 29200
29120 DEN=4:BPS=256
29200 GOSUB 29400:IF SST>127 THEN STOP
29300 POKE 768,1:POKE 4881,2:IF DEN=0 THEN POKE 4881,1
29310 RETURN
29400 TBL=ADR(TBL$):ADDR=TBL:POKE TBL+5,DEN:POKE TBL+6,INT(B
PS/256):POKE TBL+7,BPS-PEEK(TBL+6)*256:CMD$="O"
29500 POKE 768,ASC("1"):POKE 769,DRV:POKE 770,ASC(CMD$):POKE
 771,128:IF CMD$="N" THEN POKE 771,64
29510 POKE 773,INT(ADDR/256):POKE 772,ADDR-256*PEEK(773):POK
E 774,3:POKE 775,0:POKE 776,12:POKE 777,0
29520 SST=USR(ADR(CAL$)):RETURN
```

DISK DRIVE INTERFACE (via SIO)

Contributed by Bruce Blake

This article contains information and data from the MYDOS 3.05 Manual, the ATARI Technical Reference Notes, and the ARCHIVER/EDITOR User's Manual.

The physical disk drives and diskettes are external to the ATARI home computers and are normally attached to the "serial interface connector" on the right side of the computer. The software in the operating system (OS ROMs) that access the devices attached to the serial interface connector is called the "serial I/O driver" or SIO for short. The operating system uses this driver to pass all commands and information to and from the disk drive and other peripherals. Several commands were defined by ATARI to communicate with the 810 disk drive, but an extended set of commands is required to support double density functions.

The minimum set of disk drive functions are:

Device	Unit	Command	Direction	Byte Ct.	Aux. Bytes	Function
$31	Drive#	$21 !	From Drive	128/256	1 to 720	FORMAT DISK
$31	Drive#	$50 P	To Drive	128/256	1 to 720	WRITE (no verify)
$31	Drive#	$52 R	From Drive	128/256	1 to 720	READ
$31	Drive#	$53 S	From Drive	4	1 to 720	READ STATUS
$31	Drive#	$57 W	To Drive	128/256	1 to 720	WRITE (verify)

Except for the STATUS and CONFIGURE commands, the byte count is always 128 for a small sector drive, and is 128 for the first three sectors (1, 2, and 3) of a large sector drive. All other sectors on a large sector drive are 256 bytes long.

The first byte returned by the READ STATUS command is expected to indicate the sector size. If bit 5 is a 1 then the sectors are large (256 bytes), otherwise, they are small (128 bytes). Bit 3 of this byte will indicate if the disk is write protected or not. If bit 3 is a 1 then the disk is write protected. It seems that bit 4 will indicate if the disk drive is configurable or not. If bit 4 is set the drive is configurable as single or double density.

Status Request Data	First Byte
$00,$FF,$E0,$00	SD WP Off ATARI 810 Only
$08,$FF,$E0,$00	SD WP On ATARI 810 Only
$10,$FF,$E0,$00	SD WP Off
$30,$FF,$E0,$00	DD WP Off
$18,$FF,$E0,$00	SD WP On
$38,$FF,$E0,$00	DD WP On

Byte two of the floppy disk Controler status byte. A low bit indicates the error exists.

Bit	Read	Write	Notes
7	Not Ready	Not Ready	Always Clear
6	Data Error #1	Write Protect	
5	Data Error #2	Write Fault	
4	Record Not Found	Record Not Found	Sector Missing
3	CRC Error	CRC Error	
2	Lost Data	Lost Data	Should Not Happen
1	DRQ	DRQ	Always Clear
0	Busy	Busy	Always Clear

Byte three is the number of seconds required to format a disk.

Byte four is not used.

The new commands added are:

Device	Unit	Command	Direction	Byte Ct.	Aux. Bytes	Function
$31	Drive#	$4E N	From Drive	12	1 to 720	READ Configuration
$31	Drive#	$4F O	To Drive	12	1 to 720	WRITE Configuration

(21)

These commands permit reconfiguration of a disk drive on demand. These commands also support identification of a disk drive as single or double density, 5 or 8 inch and one or two sided. The individual bytes are defined as follows:

byte 0: Tracks per side (40 for a standard disk drive)
byte 1: Disk Drive Step Rate (as defined by Western Digital)

Code value	8 inch rate	5 1/4 inch rate
0	3 ms/track	6 ms/track
1	6 ms/track	12 ms/track
2	10 ms/track	20 ms/track
3	15 ms/track	30 ms/track

byte 2: Zero (high byte of sectors/track)
byte 3: Sectors/Track (18 for standard diskettes)
byte 4: Side Code (0=single sided, 1=double sided)
byte 5: Disk Type Code --
bit 2: 0=single density, 1=double density
bit 1: 0=5 1/4 inch diskette, 1=8 inch diskette drive
byte 6: High byte of Bytes/Sector (0 for ATARI 810 compatible)
byte 7: Low byte of Bytes/Sector (128 for ATARI 810)
byte 8: Translation control -- (Not valid on TRAK AT-D2)
bit 7: 1=40 trk. disk I/O on an 80 trk. drive
bit 6: Always 1 (to indicate drive present)
bit 1: 1=Handle sectors 1, 2, and 3 as full size sectors
bit 0: 1=Sectors number 0-17 (for example) not 1-18
bytes 9-11 are not used (and should be zero)

Send Configure Command Data

SD	40,0,0,18,0,0,$0,$80,1,0,0,0
DD	40,0,0,18,0,4,$1,$00,1,0,0,0

Recieve Drive Configuration Data

SD	40,3,0,18,0,0,$0,$80,$FF,$FF,$FF,$FF
DD	40,3,0,18,0,4,$1,$00,$FF,$FF,$FF,$FF

An additional requirement is that the disk drive automatically switch density if required to read sector 1. This is necessary if one is to 'BOOT' either single or double density disks.

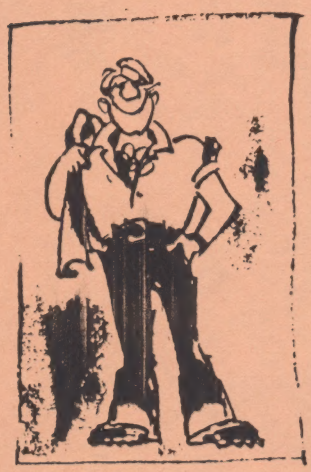
The following program will let you interrogate and configure your disk drives

```
10 REM <<<<<<<<<<<<<<>>>>>>>>>>>>>>
20 REM <<< PERCOM DISK CONTROL >>>
30 REM <<<   SAMPLE PROGRAM    >>>
40 REM <<<                     >>>
50 REM <<<     UPDATED BY      >>>
60 REM <<<     BRUCE BLAKE     >>>
70 REM <<<<<<<<<<<<<<>>>>>>>>>>>>>>
100 DIM BUF$(12):REM OPTION TABLE
110 DIM R$(1):REM RESPONSES GO HERE
120 Z=ADR("1234")
130 DATA 104,76,89,228
140 RESTORE 130:FOR I=0 TO 3
150 READ X:POKE Z+I,X:NEXT I
160 DCB=768:DRIVE=1
170 DATA 49,1,78,64,0,0,1,0,12,0,1,0
180 REM DEVICE,UNIT,COMMAND,STATUS(DIR),
190 REM BUFFER ADDR,TIMEOUT,SECTOR ADDR
200 RESTORE 170:FOR I=0 TO 11:REM BUILD THE DCB
210 READ X:POKE DCB+I,X:NEXT I
220 A=ADR(BUF$):REM INSERT BUFFER ADDRESS
230 X=INT(A/256):POKE DCB+5,X:POKE DCB+4,A-X*256
240 TRAP 250:PRINT CHR$(125):PRINT "Drive Number";:INPUT DRIVE
250 TRAP 45678:POKE DCB+1,DRIVE:REM SET DRIVE IN DCB
260 X=USR(Z):REM READ THE OPTION TABLE
270 IF PEEK(DCB+3)<>1 THEN PRINT "Drive NOT Configurable":GOSUB 590:END
280 PRINT "     Number of Tracks=";PEEK(A)
290 PRINT "     Step Rate Code=";PEEK(A+1)
300 PRINT "     Sectors/Track=";PEEK(A+3)
310 PRINT "     ";PEEK(A+4)+1;" Sided Drive"
320 IF PEEK(A+5)=0 THEN PRINT "                Single Density":GOTO 340
330 IF PEEK(A+5)=4 THEN PRINT "     Double Density"
340 PRINT "               ";PEEK(A+6)*256+PEEK(A+7);" Bytes/Sector"
360 PRINT :PRINT "Make it Single or Double density ";
370 INPUT R$
380 DATA 49,1,79,128,0,0,1,0,12,0,1,0
390 RESTORE 380:FOR I=0 TO 11:REM BUILD THE DCB
400 READ X:POKE DCB+I,X:NEXT I
410 IF R$="D" THEN 460
420 POKE A+5,0:? "Make Drive Single Density":PRINT
430 POKE A+6,0
440 POKE A+7,128
450 GOTO 490
460 POKE A+5,4:? "Make Drive Double Density":PRINT
470 POKE A+6,1
480 POKE A+7,0
490 A=ADR(BUF$):REM INSERT BUFFER ADDRESS
500 X=INT(A/256):POKE DCB+5,X:POKE DCB+4,A-X*256
510 POKE DCB+1,DRIVE:REM SET DRIVE IN DCB
520 X=USR(Z):REM UPDATE DRIVE OPTIONS
530 IF PEEK(DCB+3)<>1 THEN PRINT "Drive Update Error":GOTO 550
540 GOSUB 590
550 DATA 104,76,224,7
560 RESTORE 550:FOR I=0 TO 3
570 READ X:POKE Z+I,X:NEXT I
580 X=USR(Z):END :REM UPDATE DOS
590 DATA 49,1,83,64,0,0,1,0,4,0,1,0
600 REM DEVICE,UNIT,COMMAND,STATUS(DIR),
610 REM BUFFER ADDR,TIMEOUT,SECTOR ADDR
620 RESTORE 590:FOR I=0 TO 11:REM BUILD THE DCB
630 READ X:POKE DCB+I,X:NEXT I
640 A=ADR(BUF$):REM INSERT BUFFER ADDRESS
```

```
130 DATA 104,76,89,228
140 RESTORE 130:FOR I=0 TO 3
150 READ X:POKE Z+I,X:NEXT I
160 DCB=768:DRIVE=1
170 DATA 49,1,78,64,0,0,1,0,12,0,1,0
180 REM DEVICE,UNIT,COMMAND,STATUS(DIR),
190 REM BUFFER ADDR,TIMEOUT,SECTOR ADDR
200 RESTORE 170:FOR I=0 TO 11:REM BUILD THE DCB
210 READ X:POKE DCB+I,X:NEXT I
220 A=ADR(BUF$):REM INSERT BUFFER ADDRESS
230 X=INT(A/256):POKE DCB+5,X:POKE DCB+4,A-X*256
240 TRAP 250:PRINT CHR$(125):PRINT "Drive Number";:INPUT
DRIVE
250 TRAP 45678:POKE DCB+1,DRIVE:REM SET DRIVE IN DCB
260 X=USR(Z):REM READ THE OPTION TABLE
270 IF PEEK(DCB+3)<>1 THEN PRINT "Drive NOT
Configurable":GOSUB 590:END
280 PRINT " Number of Tracks=";PEEK(A)
```



ALBERT TENNER



ALBERT TENNER
(ELECTRONICALLY ENHANCED)

Sector Data Storing. . . .

```
0100 ;**********************************
0110 ;**A ROUTINE FOR STORING RAM ON **
0120 ;**A DISK OR FOR READING IT BACK**
0130 ;**BY ANDREW LIEBERMAN 7/10/82 **
0140 ;**********************************
0150 NUMSEC=$600 ;NUMBER OF SECTORS STIIL TO BE DONE
0160 DUNIT=$301 ;WHICH DRIVE?(1-4)
0170 DCOMD=$302 ;$52=READ, $57=WRITE
0180 DBUFLO=$304 ;POINTER FOR LO BYTE OF RAM
0190 DBUFHI=$305 ;POINTER FOR HI BYTE OF RAM
0200 DAUXLO=$30A;POINTER FOR LO BYTE OF SECTOR
0210 DAUXHI=$30B;POINTER FOR HI BYTE OF SECTOR
0220 *=$650
0230 ;THE USR COMMAND PLACES DATA ON THE STACK
0240 ; THIS PART OF THE PROGRAM PULLS THE DATA OFF AND
PUTS IT IN THE
0250 ;PROPER MEMORY LOCATIONS
0260 PLA ;WE DON'T CARE ABOUT THIS
0270 PLA
0280 STA DBUFHI
0290 PLA
0300 STA DBUFLO
0310 PLA
0320 STA DAUXHI
0330 PLA
0340 STA DAUXLO
0350 PLA ;THIS IS ASSUMED TO BE 0
0360 PLA
0370 STA NUMSEC
0380 PLA ;THIS IS ASSUMED TO BE 0
0390 PLA
0400 STA DCOMD
0410 LDA #$01 ;ASSUME DRIVE #1
0420 STA DUNIT
0430 LOOP DEC NUMSEC ;ONE LESS SECTOR TO BE DONE
0440 BMI END ;IF MINUS RESULT, LAST SECTOR WAS 0, SO
BRANCH TO END
0450 JSR $E453 ;THIS IS THE O.S. SUBROUTINE THAT DOES ALL
THE WORK
0460 CLC
0470 INC DAUXLO ;INCREMENT SECTOR POINTER
0480 BCC SKIP1 ;CHECK FOR CARRY
0490 INC DAUXHI ;THERE WAS A CARRY SO HI BYTE IS
INCREMENTED
0500 SKIP1 LDA DBUFLO ;SINCE EACH SECTOR IS $80 BYTES
LONG, THE
0510 CLC ;RAM POINTER, DBUF, MUST BE INCREMENTED BY $80
0520 ADC #$80 ;ADD $80 TO LO BYTE
0530 BVC SKIP2 ;IF IT DIDN'T OVERFLOW EVERYTHING'S O.K.
0540 INC DBUFHI ;LO BYTE OVERFLOWED, SO INCREMENT HI
BYTE
0550 SKIP2 STA DBUFLO ;DON'T FORGET TO STORE THE LO BYTE
0560 CLC ;A JUMP DONE THIS WAY MAKES THE PROGRAM
RELOCATABLE IN RAM
0570 BCC LOOP
0580 END RTS ;ALL DONE
```

4-JOYSTICK XL/XE

ACEEO - 9-86

(Reprint: A-Bug, June, 1986, Boston Computer Society)

When the Atari 1200XL appeared, one criticism was the new machine has only two joystick ports, rather than the four ports featured by the earlier 400 and 800 machines. This limitation is also present in all the newer Atari machines. Owners of these machines cannot play games intended for more than two players – until now!

The interface described in this article allows you to connect four joysticks to your XL or XE machine. With the software included, it is compatible with most programs intended for the earlier computers. The interface plugs into the joystick ports and requires no modification to your computer.

How the Interface Works

Lines 1 and 2 of joystick port 2 are programmed as outputs and used to select one of the four input jacks. The modified operating system outputs each of the four port addresses in turn. The data selectors in the interface connect the appropriate inputs to joystick port 1; the OS reads them from there. The stick values are put into the expected memory locations, so the interface is compatible with most software. Paddle inputs are connected directly from input ports 1 and 2 to the corresponding ports of the computer and are scanned normally

Building the Interface

The schematic diagram and suggested board layout appear in Figure 1. The terminal points shown in the diagram are connected to pins of the connector plugs and jacks as indicated. Parts list:

IC1: 74LS21
IC2-4: 74LS153
C1: 22 uf tantalum
C2-5: .1 uf monolithic ceramic
C6-32: .001 uf monolithic ceramic
R1-27: 220 ohm, ¼ watt
J1-4: DB9P
P1-2: DB9S

The Software

The software for the interface (Listing 1) modifies the Revision B operating system found on the Translator Disk. It changes the joystick scanning code to work with the interface. It will work with either side of the Atari Translator disk, the Fix XL from Antic, or the Home-Made Translator from Analog. The Basic program creates an AUTORUN.SYS file. Put it on a disk with DOS. This is the 4-port handler disk.

If you have the Home-Made Translator, you can make a disk for use with the interface which boots in one step. Boot the HM Translator, then the handler disk. Load the Basic program which creates the HM Translator and run it

How to Use the Interface

To run programs with the 4-joystick interface, boot your Translator (with Basic enabled or disabled, as needed), then boot the 4-port handler disk. Next insert your program disk and press Select. Plug the interface into the joystick ports of your machine, and plug joysticks or other compatible controllers into the interface

Limitations

Some programs will not work with this 4-joystick interface. Possible problems:

Reading the ports directly. This is usually done only for devices other than joysticks.

Changing any of the port lines to outputs. Devices which require this include keyboard controllers (NOT the CX85 numeric keypad), MPP modems, and printer interfaces.

Devices using the paddle inputs (paddles, CX85 keypad) will only work in the first two ports.

Any software which will not run with the OS in RAM. This is usually caused by protection schemes, not any inherent problem in the program.

Cartridge software is difficult to use with the Translator. Using a cartridge backup program which converts them to disk files will let you use them with this driver.

The interface consumes about 20 ma. This should not be a problem unless you have substantial extra power supply loads on your machine.

If you have a Ramrod XL or similar board, you could burn the modified Translator into an EPROM and install it. This should solve the problems with protected software and cartridges.

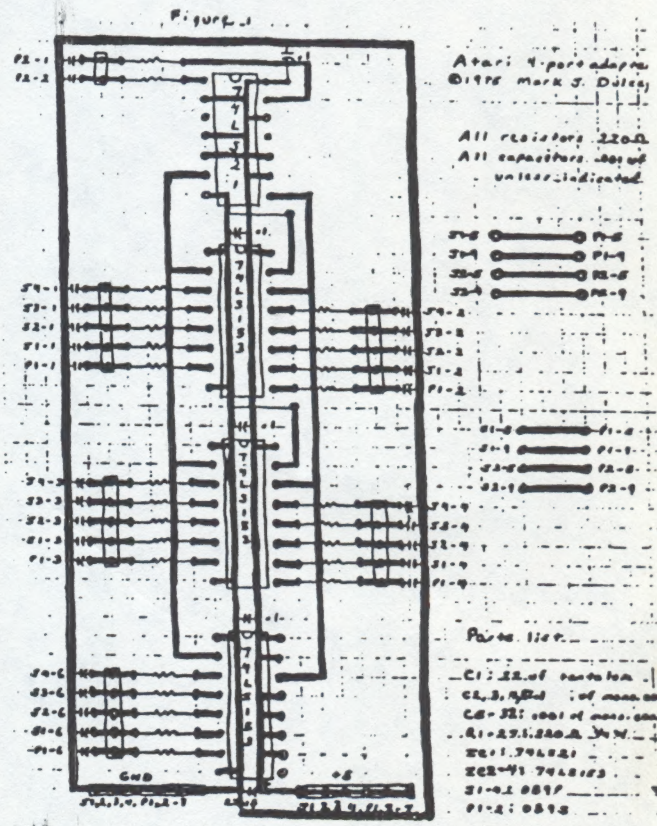
The one piece of software I've not been able to get to work with this interface is M.U.L.E. There are at least two versions of the program. The earlier one appears to read the ports directly; the later one will not run with the OS in RAM. If anyone can tell me how to use this interface with M.U.L.E., I will be very grateful!

– Mark J. Dulcey

29

LISTING 1 ©1986 MARK J. DULCEY

```
10 OPEN #1,8,0,"D:AUTORUN.SYS":TRAP 30
20 READ A:PUT #1,A:GOTO 20
30 CLOSE #1:END
1000 DATA 255,255,0,56,200,56,162,0,169,12,157,66
1010 DATA 3,32,86,228,162,0,169,3,157,66,3,169
1020 DATA 199,157,68,3,169,56,157,69,3,169,0,157
1030 DATA 74,3,169,0,157,75,3,32,86,228,162,0
1040 DATA 169,11,157,66,3,169,133,157,68,3,169,56
1050 DATA 157,69,3,169,44,157,72,3,169,0,157,73
1060 DATA 3,32,86,228,169,5,205,31,208,208,251,120
1070 DATA 169,0,141,14,212,141,14,210,169,48,141,227
1080 DATA 238,162,49,189,185,56,157,115,232,
1090 DATA 76,119,228,162,3,138,10,10,10,10,1
1100 DATA 211,234,234,234,234,234,234,234,234,23
1110 DATA 173,0,211,41,15,157,128,2,173,16,2
1120 DATA 132,2,189,0,210,157,112,2,189,4,21
1130 DATA 116,2,202,16,208,125,133,78,111,11
1140 DATA 112,111,114,116,32,97,100,97,112,1
1150 DATA 32,108,114,105,110,101,114,133,133
1160 DATA 68,114,101,115,115,32,83,69,76,69,
1170 DATA 133,69,58,226,2,227,2,0,36,224,2,2
1180 DATA 2,0,36
```



Reprint from Page 6 of England.

SONAR SEARCH

Ron Smith, Cheshire

Seek...locate...destory! Sonar Search is a submarine hunt game in the classic style of seeking targets by deduction and logic. You are presented with a grid and by using the joystick have to place a cursor in the position that you think the enemy submarine is located. You will be given a number that indicateds how far from the target you are and must then

Hi SAUG 4/86

THE HARD(WARE) FACTS

(55)

EVERYTHING YOUR WANTED TO KNOW ABOUT YOUR ATARI BUT WERE AFRAID TO ASK (PART 1)

We are pleased to welcome John J. Smith to our pages with a regular column on hardware. John's articles will feature a mixture of hard facts, ideas and projects. You will be shown inside the 800 and find out what is inside a cartridge among many other things in future Hard(ware) Facts.

JOYSTICK PORTS

The first thing most people seem to want to know is what you can plug into the I/O ports on the front of the Atari 800. The answer is many things but in order to do so, especially if you build something yourself, you will need to know what the pin connections are. Figure 1 shows the pin connections of port 1 and the remaining ports are the same. It is important to note that the connections shown are exactly as you see them when sitting in front of the computer. The connections on the computer are plugs with pins (male) and if you want to connect something you will need female sockets. These are called 9 way D type connectors and manufacturers seem to call them DB9S connectors. You will most probably get them from your local shop, if you have one, or by mail order and everyone seems to have their own part numbers. One source of supply is Maplin Electronics, P.O.Box 3, Rayleigh, Essex and their part number for the 9 way sockets is RK61R. I suggest that you also use covers to hide the wires and solder connections and the Maplin part number is RK 62S. These are sometimes known as 'hoods'.

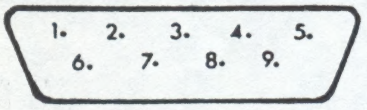


Figure 1 - controller port connections

Pin 1 Joystick - forward
Pin 2 Joystick - back
Pin 3 Joystick - left
Pin 4 Joystick - right
Pin 5 B paddle (potentiometer) input
Pin 6 Trigger input
Pin 7 5 volts available
Pin 8 Ground (Earth)
Pin 9 A paddle (potentiometer) input

Now that you know what the I/O port connections are maybe you want to build something to plug in. Let's start with something simple as even I have not yet figured out how to build a trackball! When I do I will let you have details. How about a joystick? At first this seems a good idea but if you are a real beginner, even this can be a challenge as you have to get the lever to pivot in all directions without falling apart. No, for your first project I would suggest a simple push-button cursor control to provide similar controls to a joystick i.e. Up, Down, Left, Right and Fire. Five seperate push buttons are required and the wiring diagram for these is shown in Figure 2.

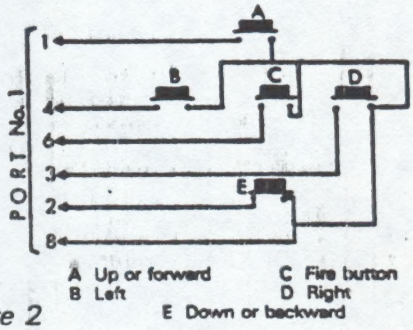


Figure 2

Note that pin 8 is common to all five push buttons and that pins 5,7 & 9 are unused so that you only need a six way cable for wiring. One possible suggestion for mounting is to use a small handy sized box and mount the buttons as shown in figure 3. A plastic ring can cover buttons A, B, D and E so that a rocking action can be used. How you achieve the final design is up to you!

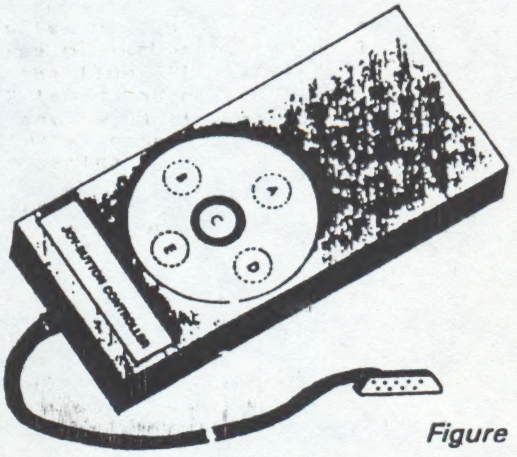
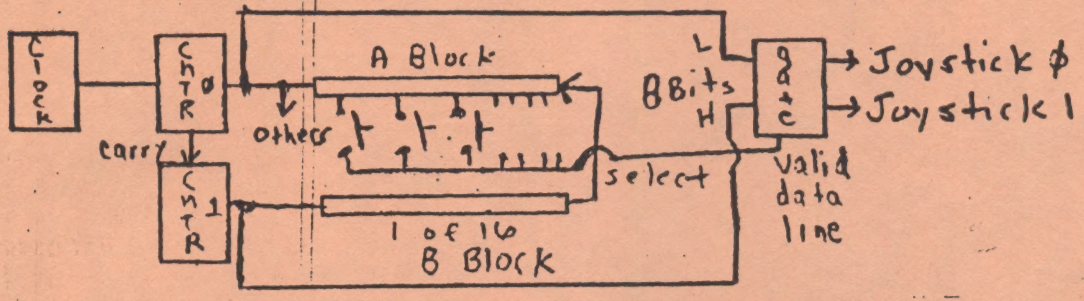


Figure 3

the Hobby Shop
by Rick Detlefsen

Last time I talked about using the joystick ports for reading the switches of the joystick and keypad. I have had inquiries as to how to do it so this month I will present a block diagram on this. Next month I will present the actual circuit to do it.



Basically you set up a clock circuit to scan each switch. Each switch is an 'A' circuit. Each sixteen switches is a 'B' block. Each 256 switches is a 'C' block. The result of the scanning is placed in the joystick port if a valid switch is pressed. Switch number zero is not used.

As the counter increments, a voltage is applied to each switch. If a switch is pressed, the circuit then assumes that the current clock number is to be read by the port. In the future, I will ennance this by using switch 255 for a special purpose.

Now then, to use the ports for output, just follow these steps:A(B)- POKE 54018,56(54019,56)-I/O control, POKE 54016,255(54017,255)-set up for output, POKE 54018,60(54019,60)-the following is data, POKE 54016,DATA(54017,DATA). To change to input-follow the above only instead of POKE 54016,255(54107) use POKE 54016,0(54017,0).

The block diagram below shows the procedure. Each four bits of the port is used to drive a 1-of-16 decoder. Only one led can be lit at a time, althought high speed may cause more than one to look lit.

Next month, I will start to present the actual circuits. BYE.

